



... for a brighter future



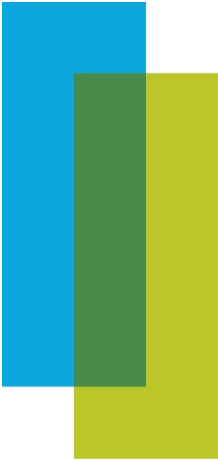
UChicago ►
Argonne_{LLC}



A U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC

TotalView Source Code Debugger

Ed Hinkel



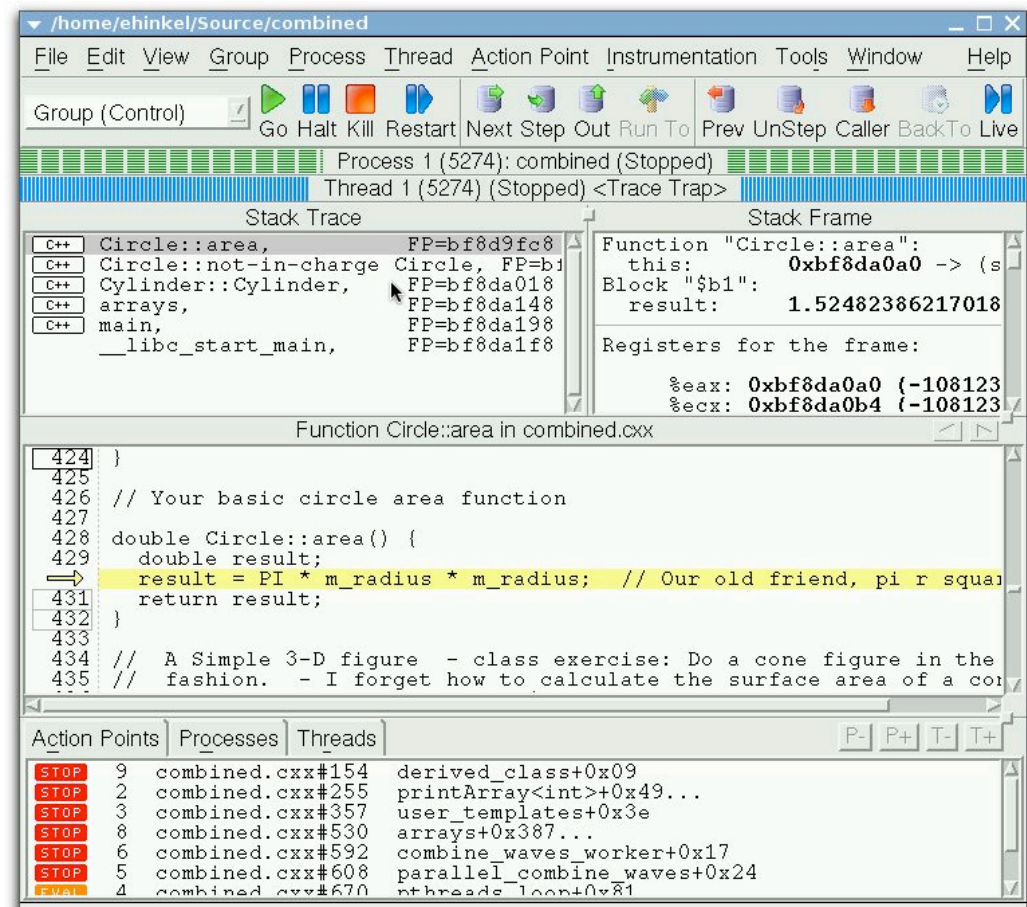
Agenda

- **Introduction**
- **Navigation and Control**
- **Parallel Debugging**
- **Memory Debugging**
- **What's New**

What is TotalView?

A comprehensive debugging solution for demanding parallel and multi-core applications

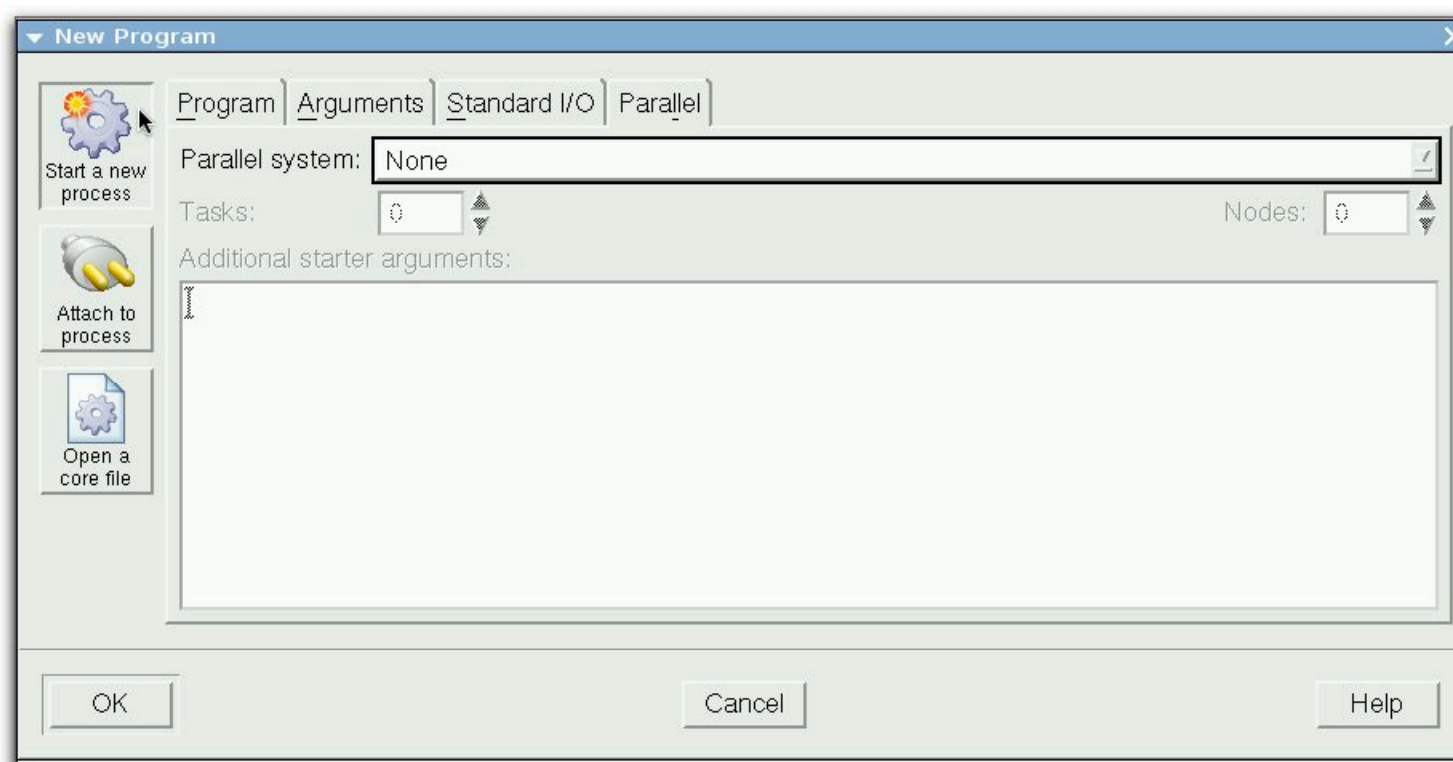
- Wide compiler & platform support
 - C, C++, Fortran 77 & 90, UPC
 - Unix, Linux, OS X
- Handles Concurrency
 - Multi-threaded Debugging
 - Parallel Debugging
 - MPI, PVM, Others
 - Remote and Client/Server Debugging
- Integrated Memory Debugging
- Reverse Debugging available
 - ReplayEngine add on
- Supports a Variety of Usage Models
 - Powerful and Easy GUI
 - Visualization
 - CLI for Scripting
 - Long Distance Remote Debugging
 - Unattended Batch Debugging



Supported Compilers and Architectures

- **Platform Support**
 - Linux x86, x86-64, ia64, Power
 - Mac Power and Intel
 - Solaris Sparc and AMD64
 - AIX, Tru64, IRIX, HP-UX ia64
 - Cray X1, XT3, XT4, IBM BGL, BGP,
 - Cell, SiCortex
- **Languages / Compilers**
 - C/C++, Fortran, UPC, Assembly
 - Many Commercial & Open Source Compilers
- **Parallel Environments**
 - MPI
 - (MPICH1& 2
 - LAM
 - Open MPI,
 - poe, MPT, Quadrics, MVAPICH, & many others)
 - UPC

Starting TotalView



Open a Core File

Starting TotalView

Via Command Line

Normal

```
totalview [ tv_args ] prog_name [-a prog_args ]
```

Attach to running program

```
totalview [ tv_args ] prog_name -pid PID# [-a prog_args ]
```

Attach to remote process

```
totalview [ tv_args ] prog_name -remote name [-a prog_args ]
```

Attach to a core file

```
totalview [ tv_args ] prog_name corefile_name [ -a prog_args ]
```

Preparing for debugging on Bluegene

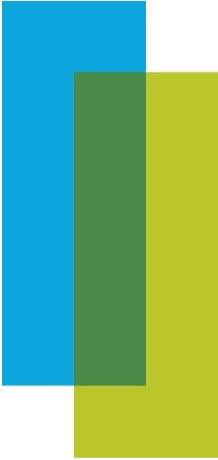
- **TotalView needs debug symbols**
 - Add '-g' to your compile line
 - If bug can be reproduced without optimization, remove any optimization flags
 - Note: GNU Bluegene/P Compiler Wrappers (mpicc, mpicxx, and mpif77) force -O3 into compilation.



Starting TotalView on ALCF Bluegene



- Submitting a job for debugging using isub...
the following is how a user would start a 64-node job:
- `isub -q prod-devel -t 60 -n 64 --run totalview mpirun -a -np 64 a.out`
- or else
- `isub -q prod-devel -t 60 -n 64`
- [wait for prompt]
- `totalview mpirun -a -np 64 a.out` You start of debugging 'mpirun'
 - Typically the first thing you would do is hit the “GO” button.
- **TotalView will tell you when the application has been halted**
 - All processes will be halted after executing exactly one instruction.

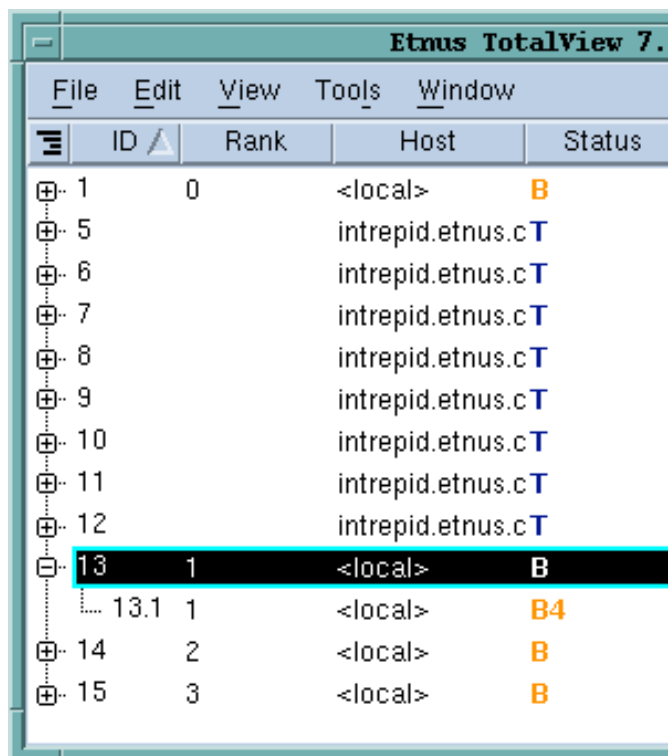


Process Control & Navigation

Interface Concepts

Root Window

- State of all processes being debugged
- Process and Thread status
- Instant navigation access
- Sort and aggregate by status

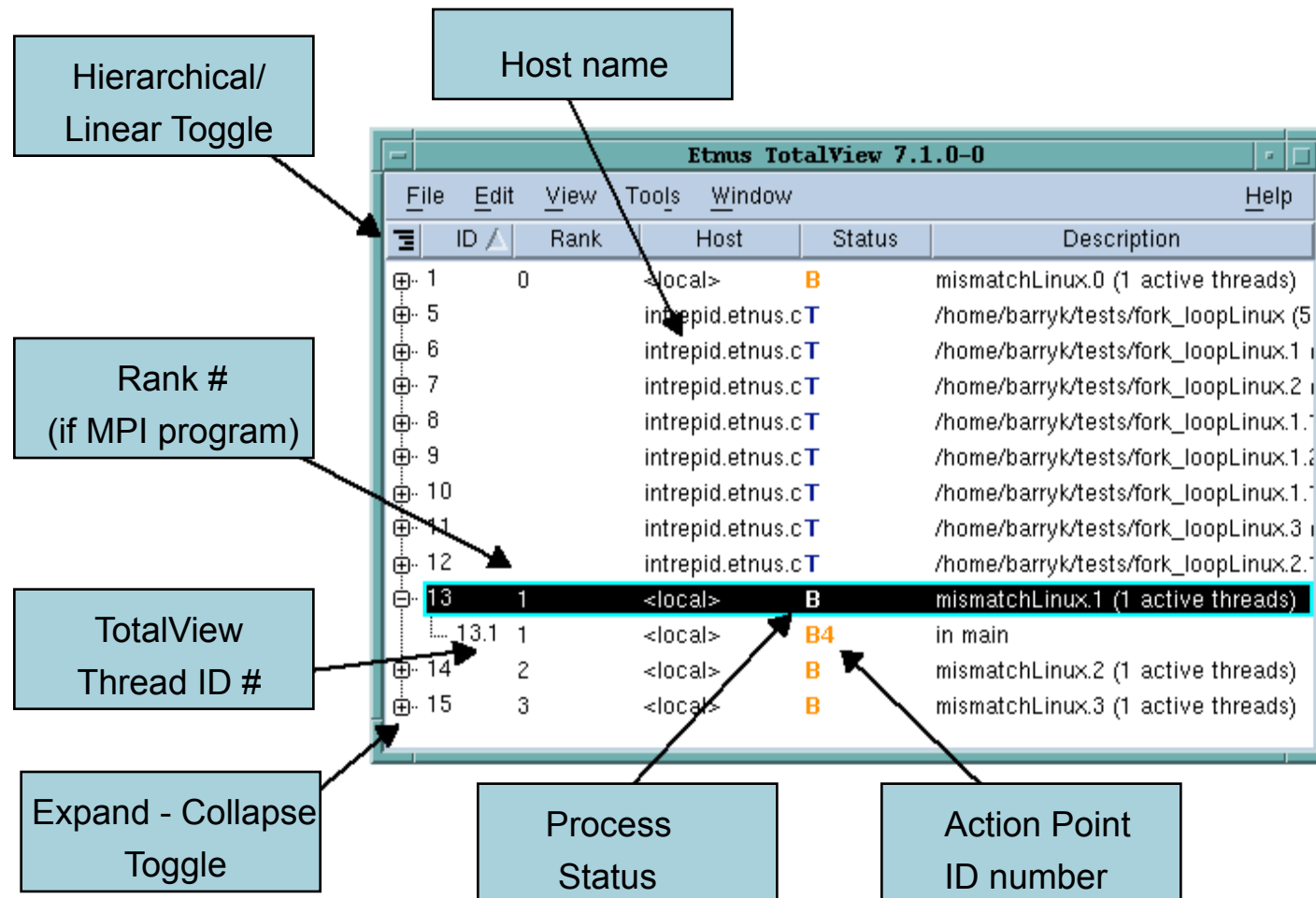


| | ID | Rank | Host | Status |
|---|------|------|------------------|--------|
| ⊕ | 1 | 0 | <local> | B |
| ⊕ | 5 | | intrepid.etnus.c | T |
| ⊕ | 6 | | intrepid.etnus.c | T |
| ⊕ | 7 | | intrepid.etnus.c | T |
| ⊕ | 8 | | intrepid.etnus.c | T |
| ⊕ | 9 | | intrepid.etnus.c | T |
| ⊕ | 10 | | intrepid.etnus.c | T |
| ⊕ | 11 | | intrepid.etnus.c | T |
| ⊕ | 12 | | intrepid.etnus.c | T |
| ⊖ | 13 | 1 | <local> | B |
| ⊕ | 13.1 | 1 | <local> | B4 |
| ⊕ | 14 | 2 | <local> | B |
| ⊕ | 15 | 3 | <local> | B |

► Status Info

- T = stopped
- B = Breakpoint
- E = Error
- W = Watchpoint
- R = Running
- M = Mixed
- H = Held

TotalView Root Window



- Dive to refocus
- Dive in new window to get a second process window

Process Window Overview

Stack Trace Pane

Toolbar

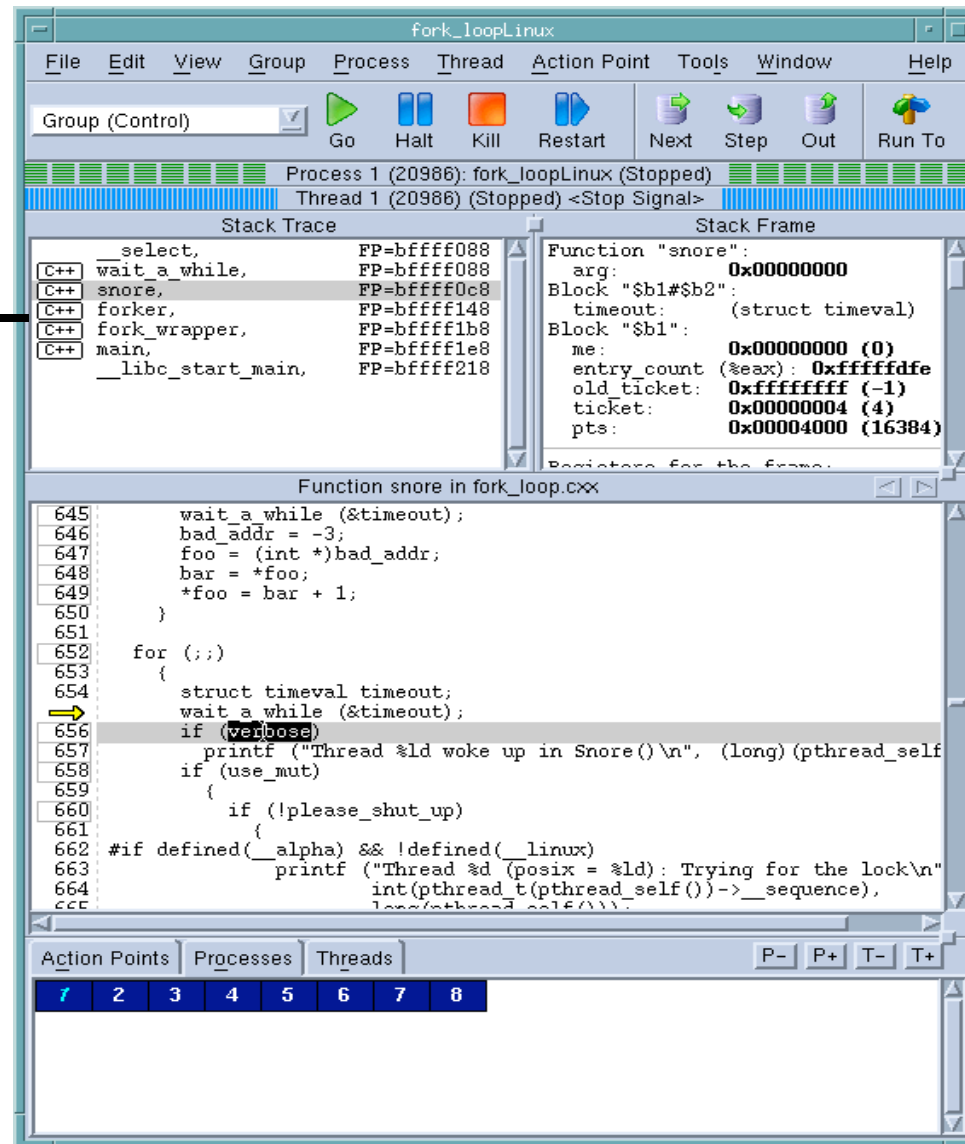
Stack Frame Pane

Source Pane

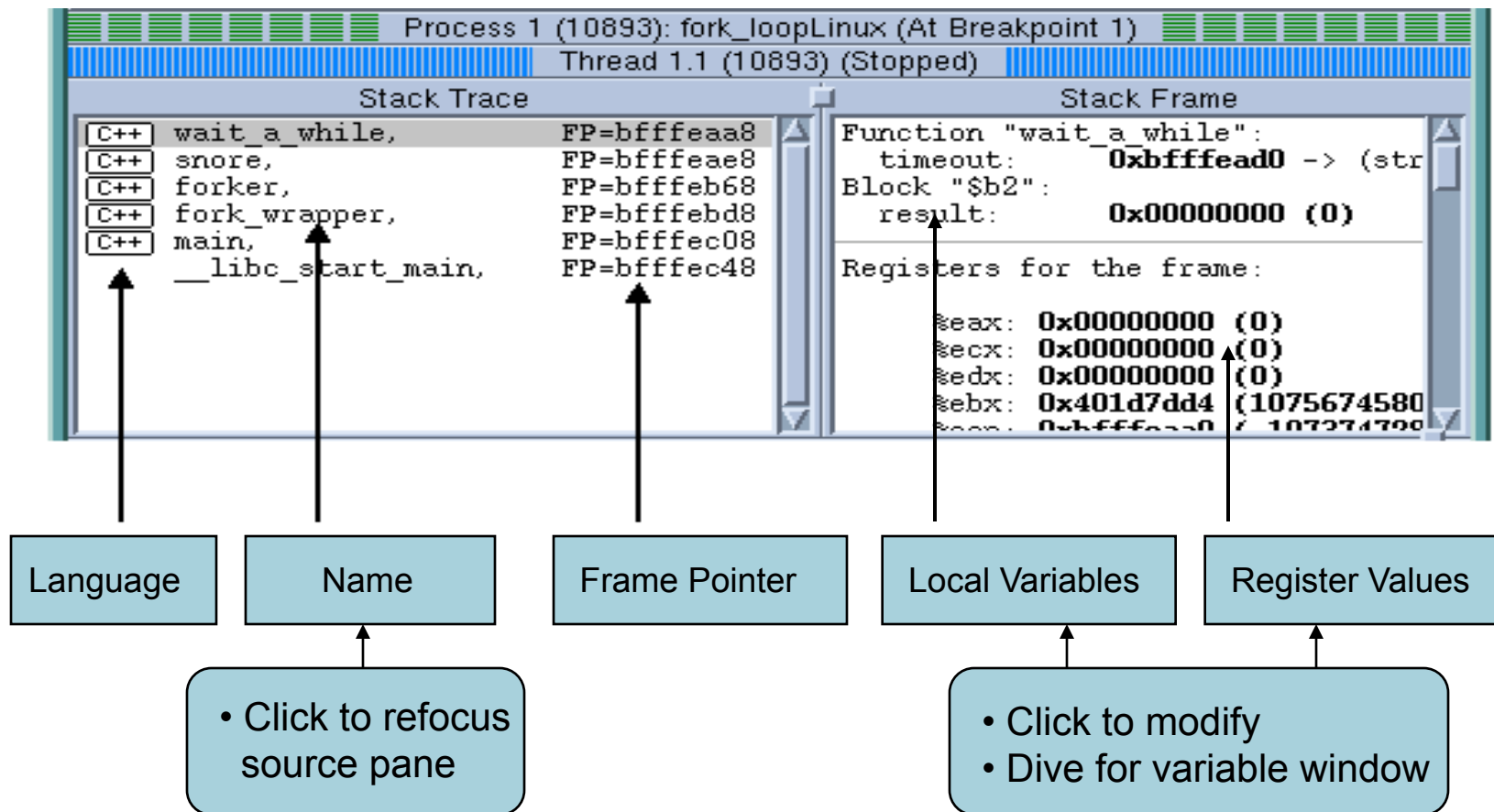
Tabbed Area

Provides detailed
state of one process,
or a single thread
within a process

A single point of
control for the
process and other
related processes

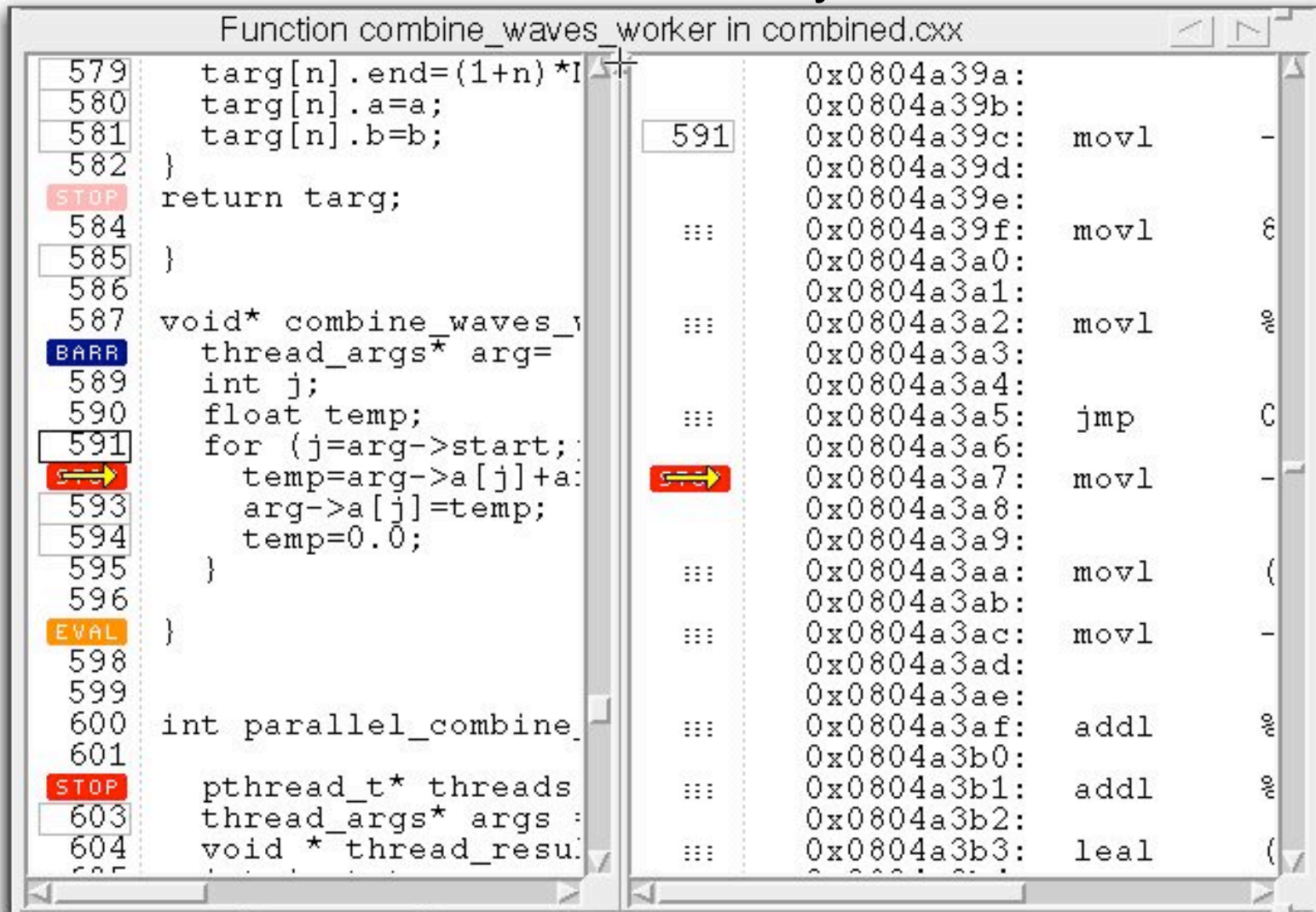


Stack Trace and Stack Frame Panes



Source Code Pane

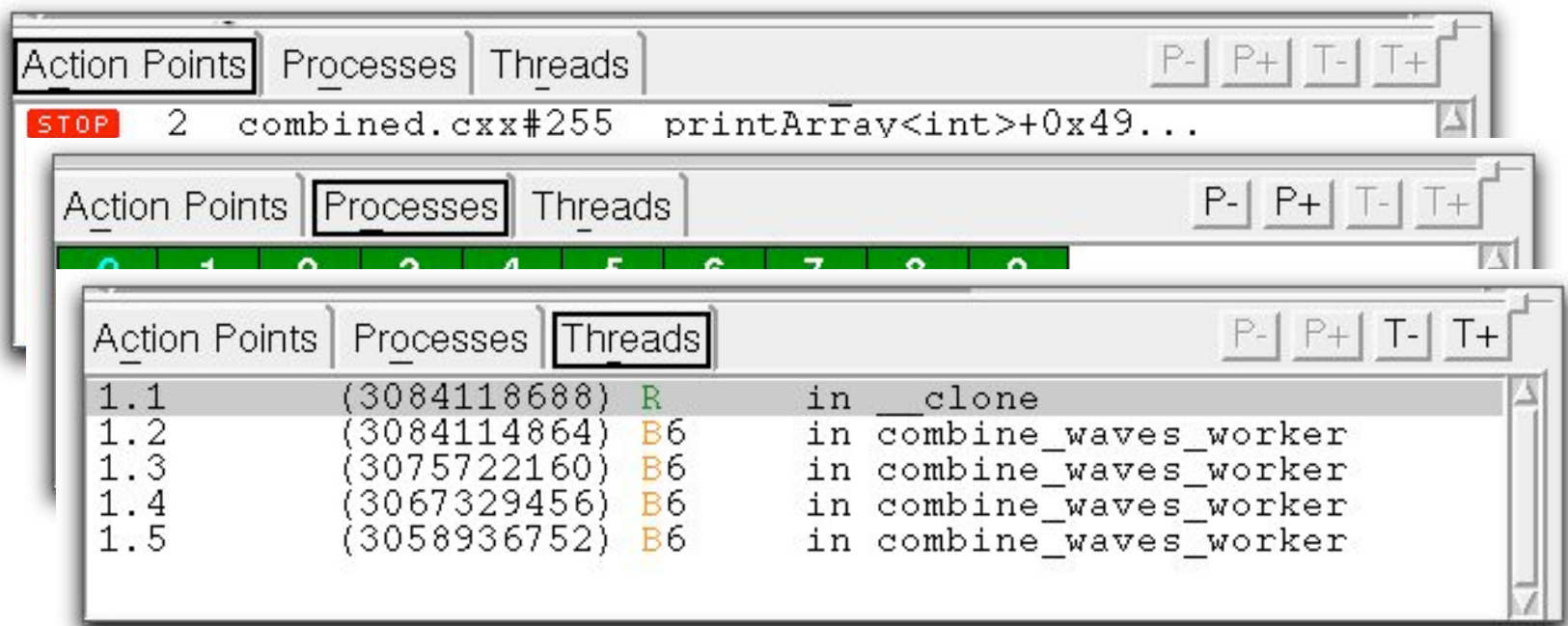
View as Source - or Assembly - or Both!



The screenshot displays the 'Source Code Pane' in TotalView, showing a comparison between C source code and its corresponding assembly code. The window title is 'Function combine_waves_worker in combined.cxx'. The left pane shows the C source code, and the right pane shows the assembly code. A red double-headed arrow highlights the correspondence between line 591 in the source code and the assembly code.

| Source Code (C) | Assembly Code |
|---------------------------|------------------------|
| 579 targ[n].end=(1+n)*I | 0x0804a39a: |
| 580 targ[n].a=a; | 0x0804a39b: |
| 581 targ[n].b=b; | 591 0x0804a39c: movl - |
| 582 } | 0x0804a39d: |
| STOP return targ; | 0x0804a39e: |
| 584 | ::: 0x0804a39f: movl 8 |
| 585 } | 0x0804a3a0: |
| 586 | 0x0804a3a1: |
| 587 void* combine_waves_ | ::: 0x0804a3a2: movl % |
| BARR thread_args* arg= | 0x0804a3a3: |
| 589 int j; | 0x0804a3a4: |
| 590 float temp; | ::: 0x0804a3a5: jmp C |
| 591 for (j=arg->start; | 0x0804a3a6: |
| temp=arg->a[j]+a; | 591 0x0804a3a7: movl - |
| 593 arg->a[j]=temp; | 0x0804a3a8: |
| 594 temp=0.0; | 0x0804a3a9: |
| 595 } | ::: 0x0804a3aa: movl (|
| 596 | 0x0804a3ab: |
| EVAL } | ::: 0x0804a3ac: movl - |
| 598 | 0x0804a3ad: |
| 599 | 0x0804a3ae: |
| 600 int parallel_combine_ | ::: 0x0804a3af: addl % |
| 601 | 0x0804a3b0: |
| STOP pthread_t* threads | ::: 0x0804a3b1: addl % |
| 603 thread_args* args : | 0x0804a3b2: |
| 604 void * thread_resu | ::: 0x0804a3b3: leal (|

Tabbed Pane



Action Points Tab
 all currently defined
 action points

Processes Tab
 all current
 processes

Threads Tab:
 all current threads,
 ID's, Status

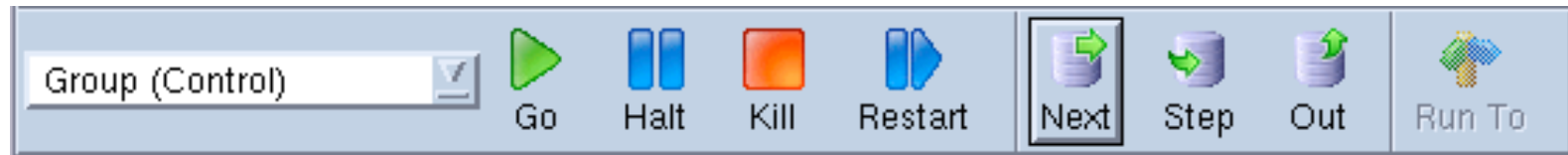
Process Status

Process/Thread status is available at a glance, in both the Process and Root Windows

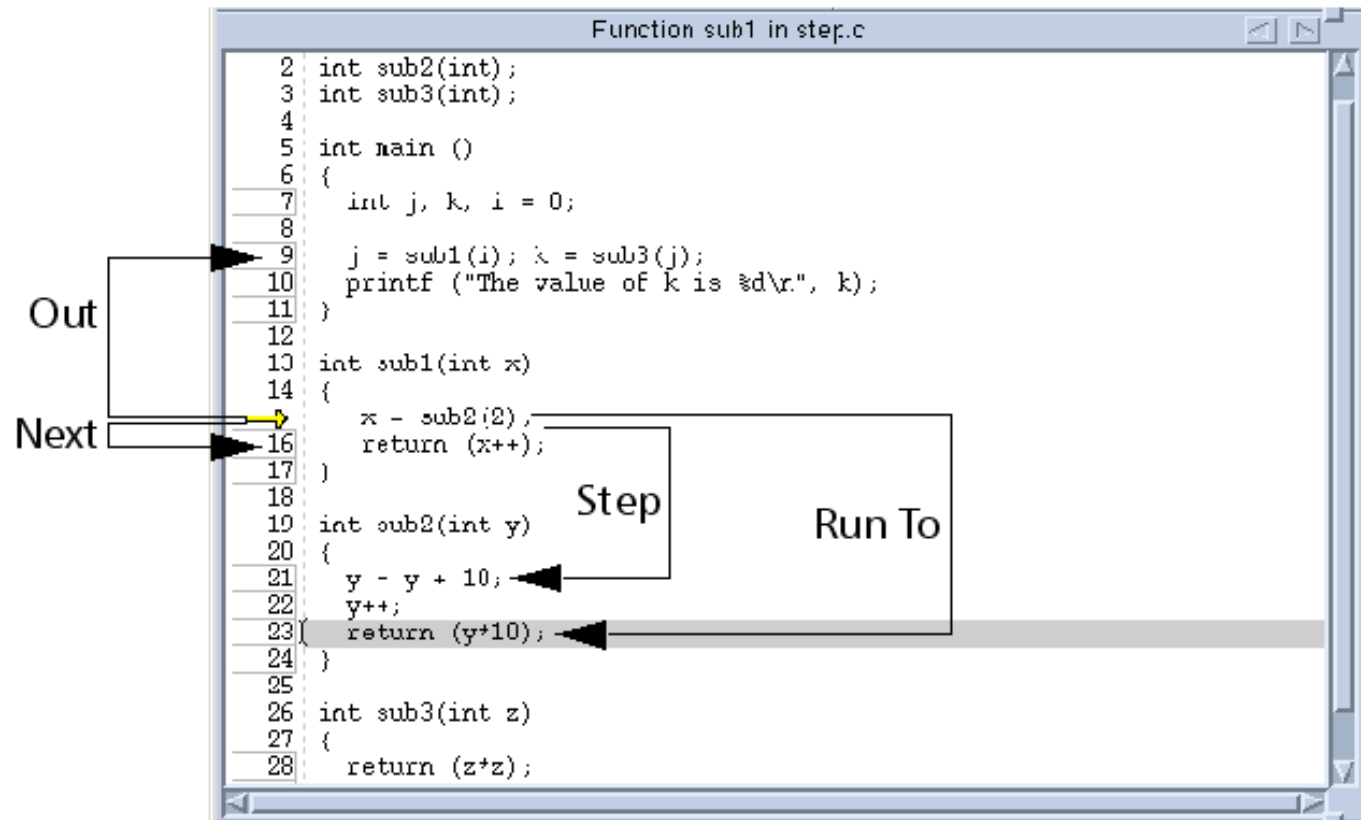
The screenshot displays the Etnus TotalView 6X.8.0-3 interface. The top window, titled 'fork_loopLinux', shows a process control bar with a red box highlighting the status of 'Process 1 (20986): fork_loopLinux (Stopped)' and 'Thread 1 (20986) (Stopped) <Stop Signal>'. Below this, a stack trace and stack frame are visible. The bottom window, titled 'Etnus TotalView 6X.8.0-3', shows a table of processes and threads. A red box highlights the first thread, '1.1', which is in the 'wait_a_while' state.

| ID | Rank | Host | Status | Description |
|-----|------|---------|--------|--|
| 1 | | <local> | B | /home/barryk/tests/fork_loopLinux (5 act |
| 1.1 | | <local> | T | in wait_a_while |
| 1.2 | | <local> | T | in wait_a_while |
| 1.3 | | <local> | T | in wait_a_while |
| 1.4 | | <local> | B1 | in wait_a_while |
| 2 | | <local> | B | /home/barryk/tests/fork_loopLinux.1 (5 a |
| 3 | | <local> | B | /home/barryk/tests/fork_loopLinux.2 (5 a |
| 4 | | <local> | B | /home/barryk/tests/fork_loopLinux.1.1 (5 |
| 5 | | <local> | B | /home/barryk/tests/fork_loopLinux.1.2 (5 |
| 6 | | <local> | B | /home/barryk/tests/fork_loopLinux.1.1.1 |
| 7 | | <local> | B | /home/barryk/tests/fork_loopLinux.2.1 (5 |
| 8 | | <local> | B | /home/barryk/tests/fork_loopLinux.3 (5 a |

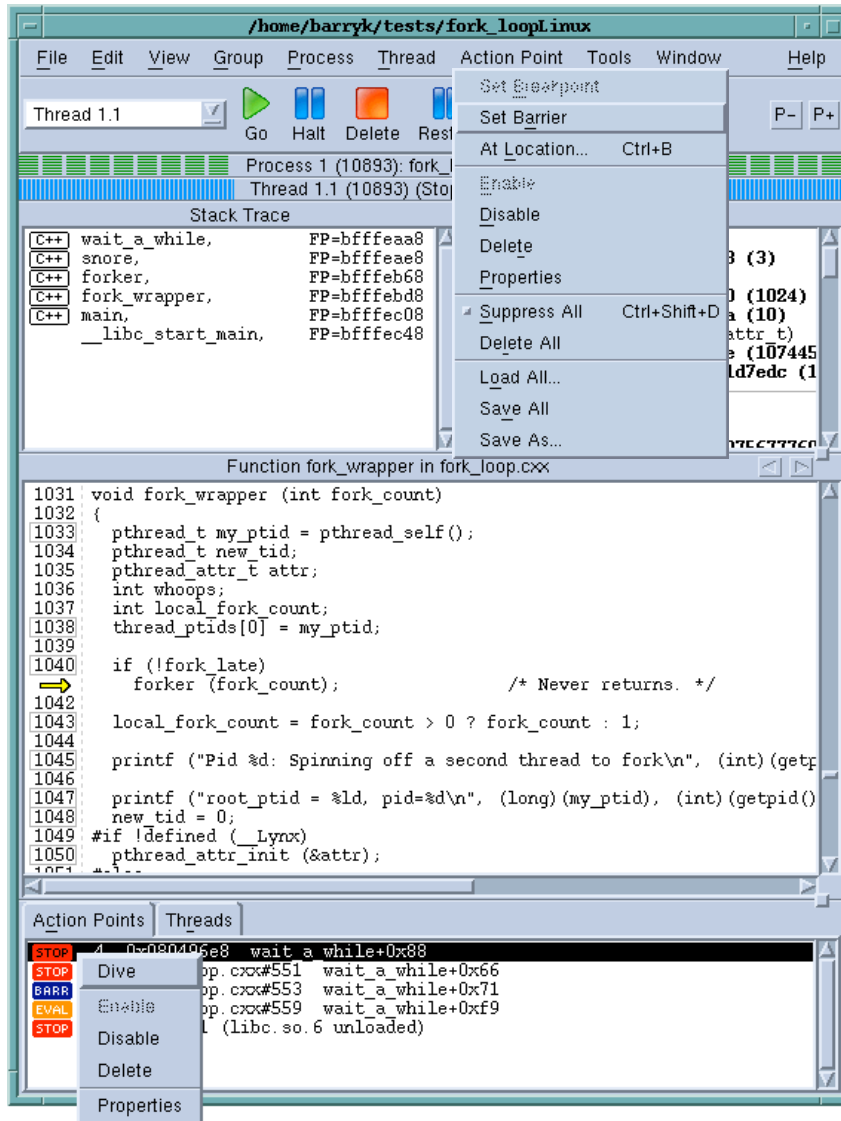
Stepping Commands



Based on
PC location



Action Points



Breakpoints

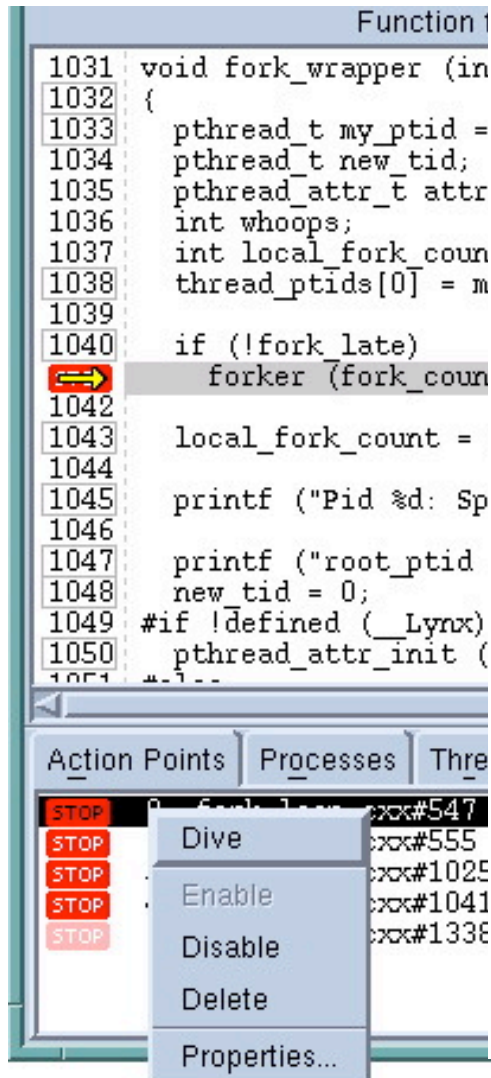
Barrier Points

Conditional Breakpoints

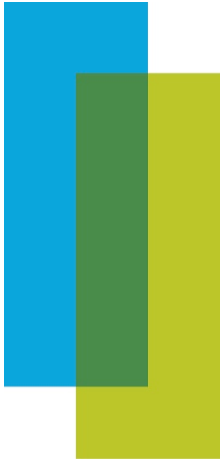
Evaluation Points

Watchpoints

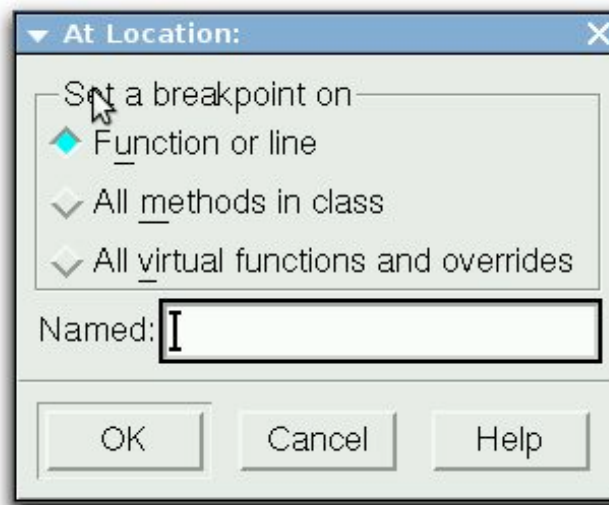
Setting Breakpoints



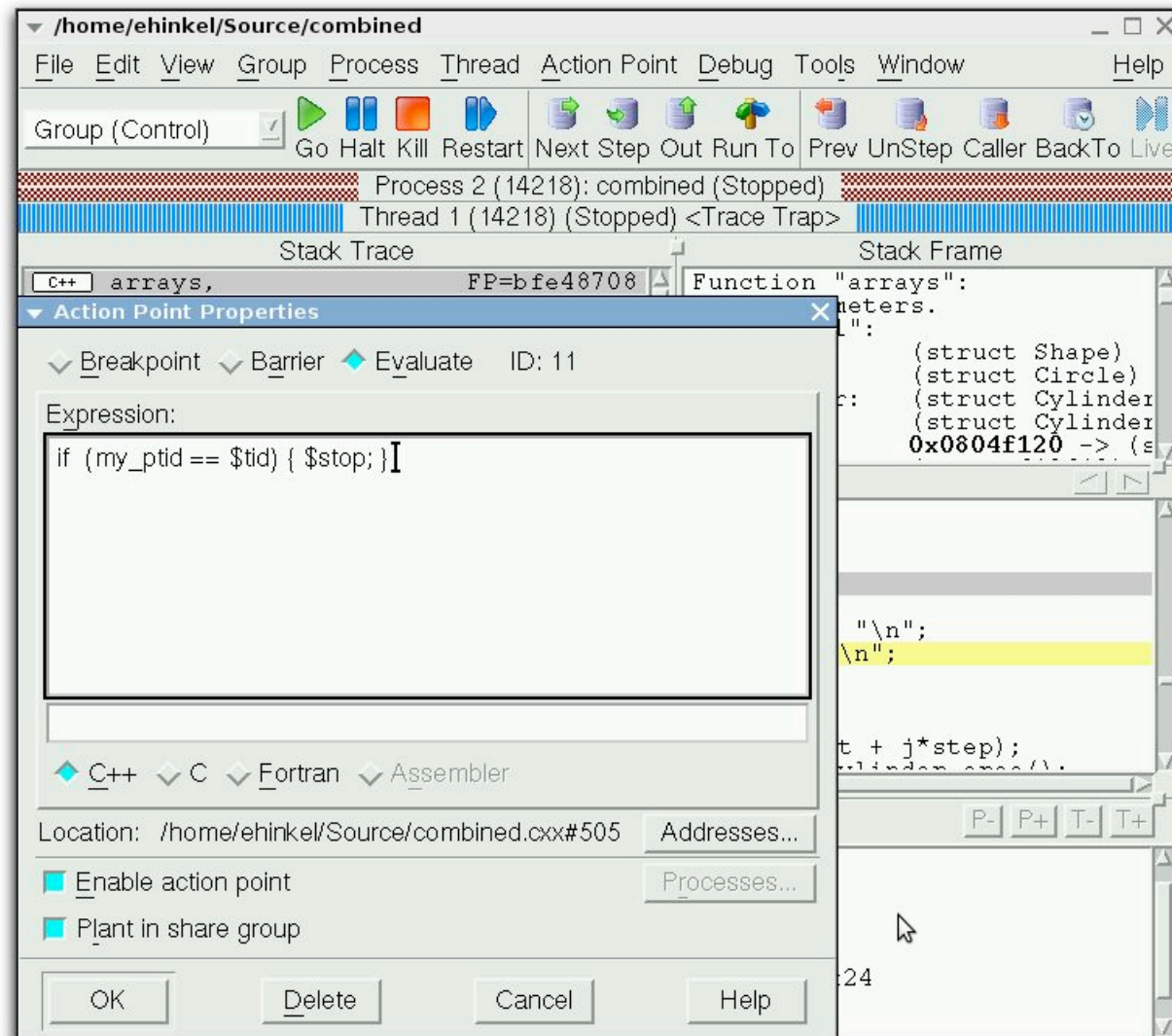
- **Setting action points**
 - Single-click line number
- **Deleting action points**
 - Single-click action point line
- **Disabling action points**
 - Single-click in Action Points Tab Pane
- **Optional contextual menu access for all functions**
- **Action Points Tab**
 - Lists all action points
 - Dive on an action point to focus it in source pane
- **Action point properties**
 - In Context menu
- **Saving all action points**
 - Action Point > Save All



Setting Breakpoints



Conditional Breakpoint



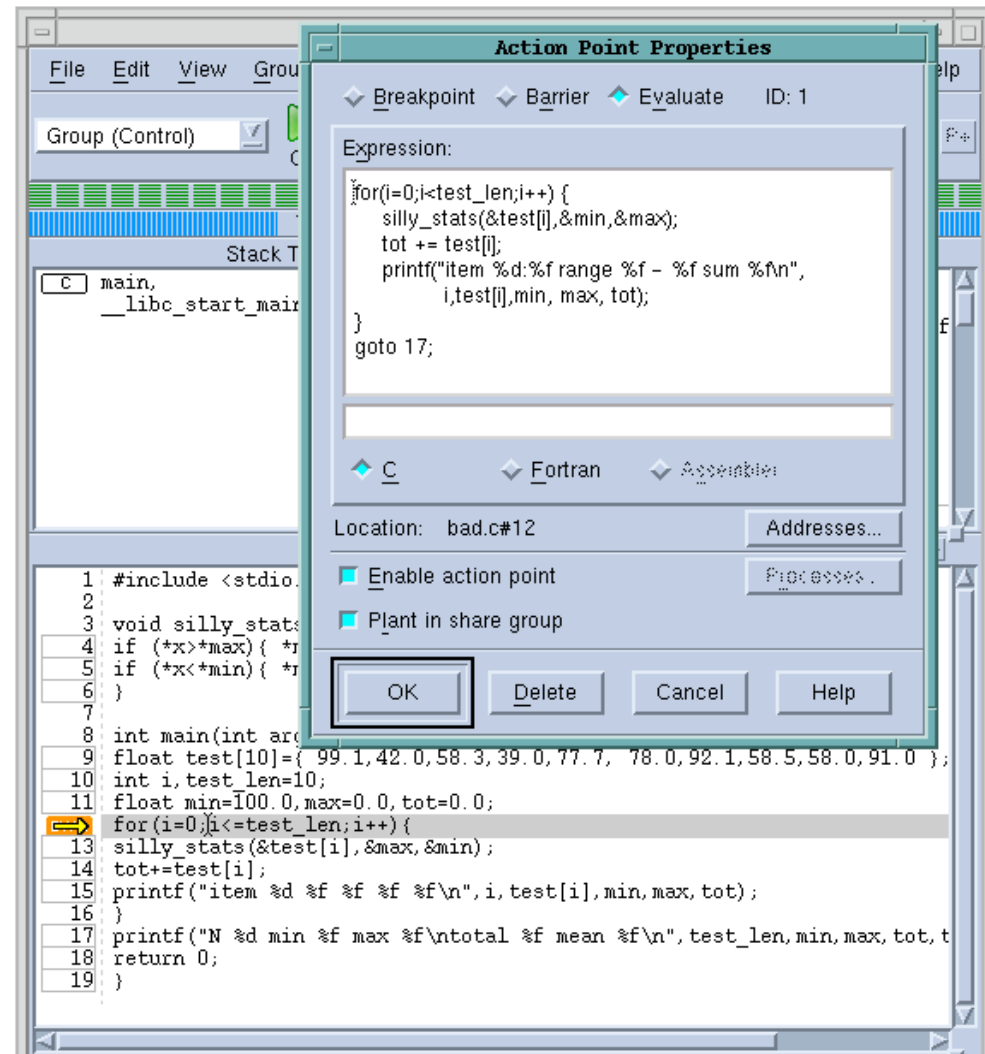
Evaluation Breakpoint...

Test Fixes on the Fly!

- Test small source code patches
- Call functions
- Set variables
- Test conditions
- C/C++ or Fortran
- Can't use C++ constructors
- Use program variables
- Can't modify variables or call functions with replay engine

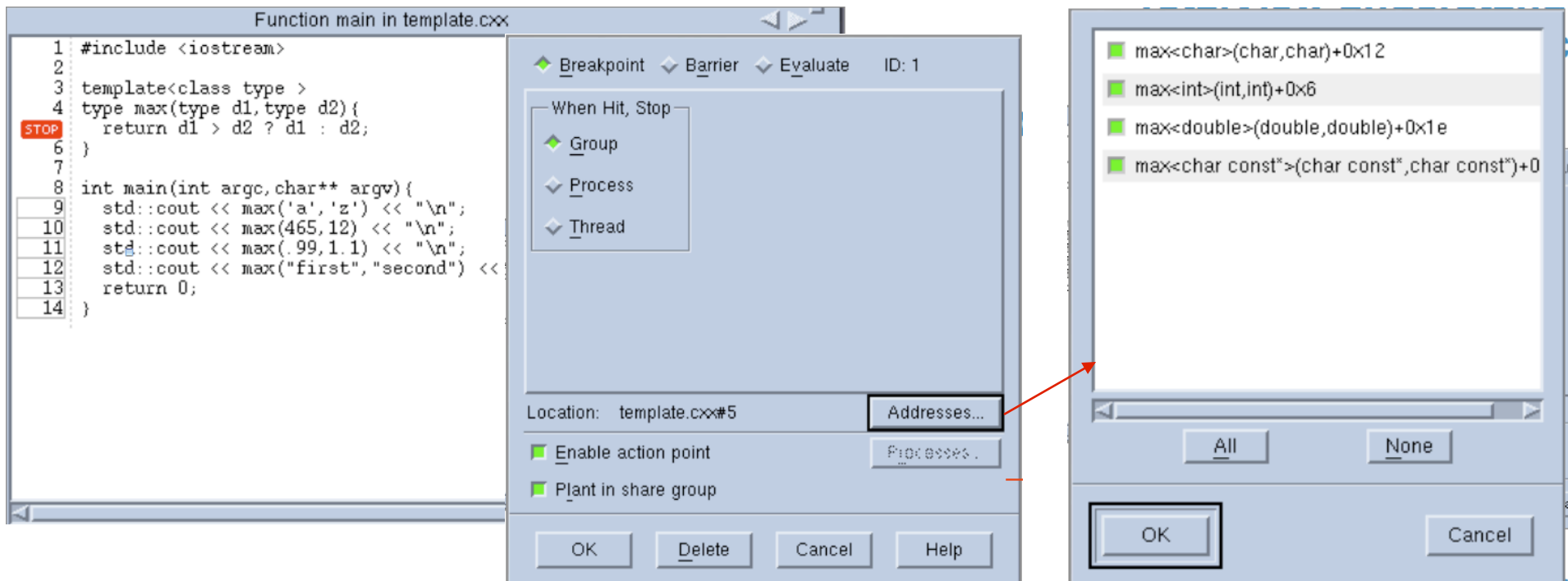
```

item 0:99.099998 range 99.099998 - 99.099998 sum 99.099998
item 1:42.000000 range 42.000000 - 99.099998 sum 141.100006
item 2:58.299999 range 42.000000 - 99.099998 sum 199.400009
item 3:39.000000 range 39.000000 - 99.099998 sum 238.400009
item 4:77.699997 range 39.000000 - 99.099998 sum 316.100006
item 5:78.000000 range 39.000000 - 99.099998 sum 394.100006
item 6:92.099998 range 39.000000 - 99.099998 sum 486.200012
item 7:58.500000 range 39.000000 - 99.099998 sum 544.700012
item 8:58.000000 range 39.000000 - 99.099998 sum 602.700012
item 9:91.000000 range 39.000000 - 99.099998 sum 693.700012
N 10 min 39.000000 max 99.099998
total 693.700012 mean 69.370001
    
```



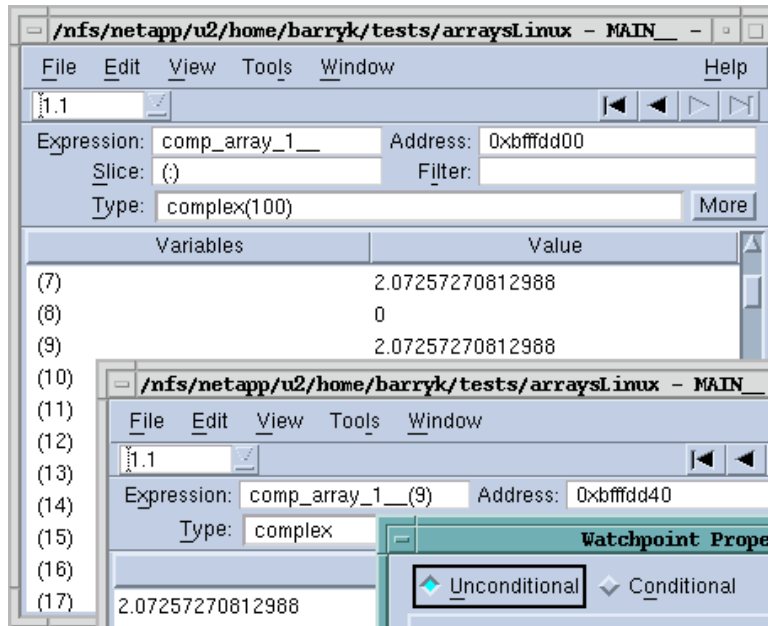
Setting Breakpoints With C++ Templates

TotalView understands C++ templates and gives you a choice ...



- 23 Boxes with solid lines around line numbers indicate code that exists at more than one location.

Watchpoints



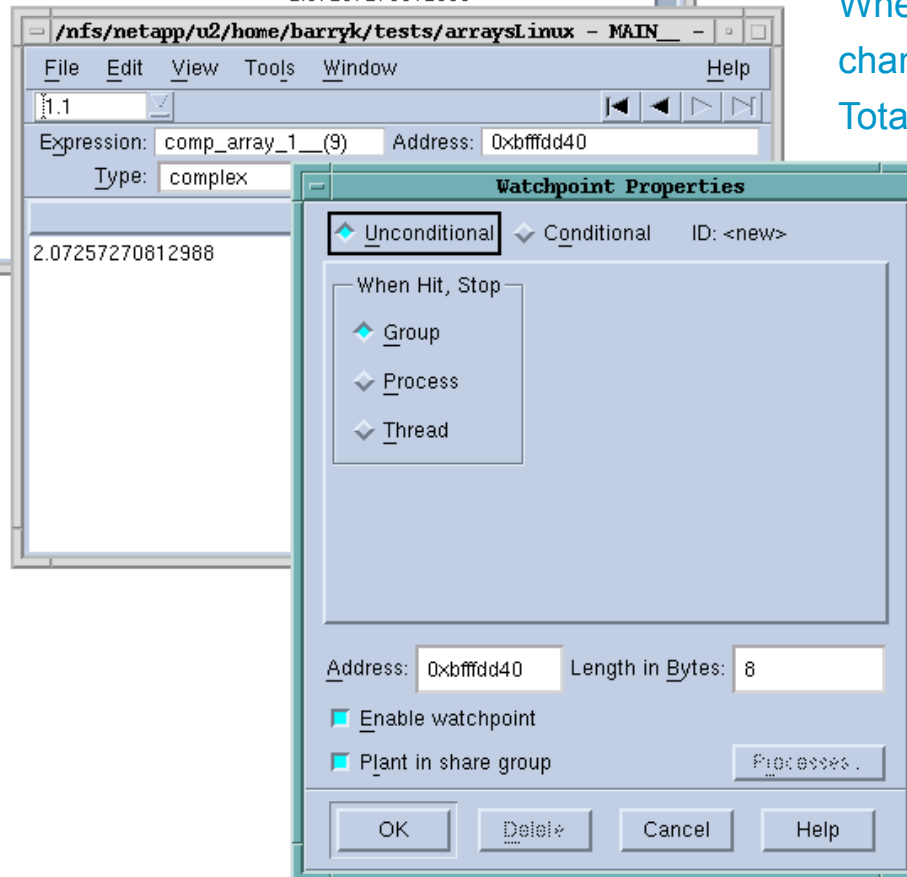
Watchpoints are set on a fixed memory region

Use *Tools > Watchpoint* from a Variable Window

or

From source pane with contextual menu

When the contents of watched memory change, the watchpoint is triggered and TotalView stops the program.



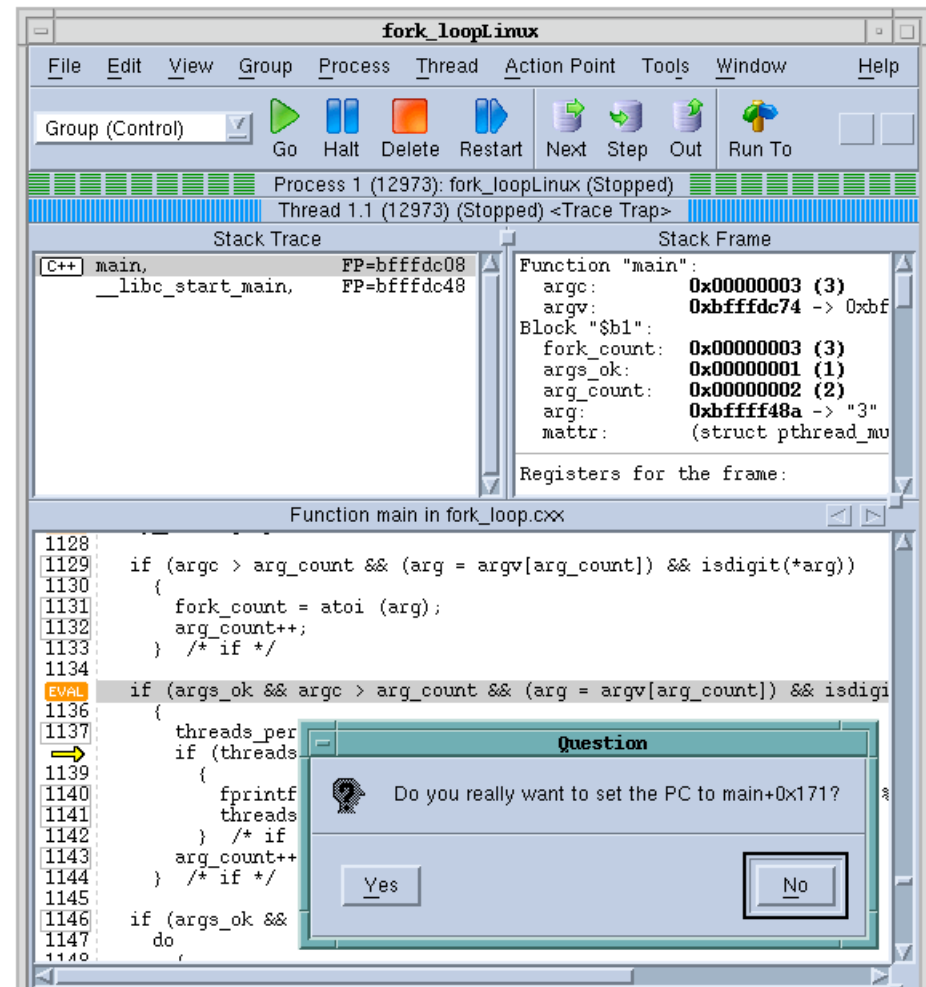
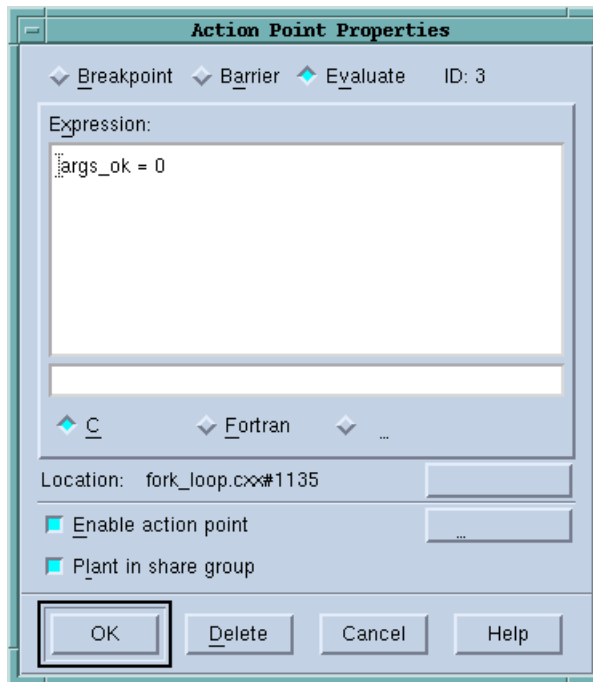
Watchpoints are not set on a variable. You need to be aware of the variable scope.

Watchpoints can be conditional or unconditional

TV variables *\$newval* and *\$oldval* can be used in the conditional expression

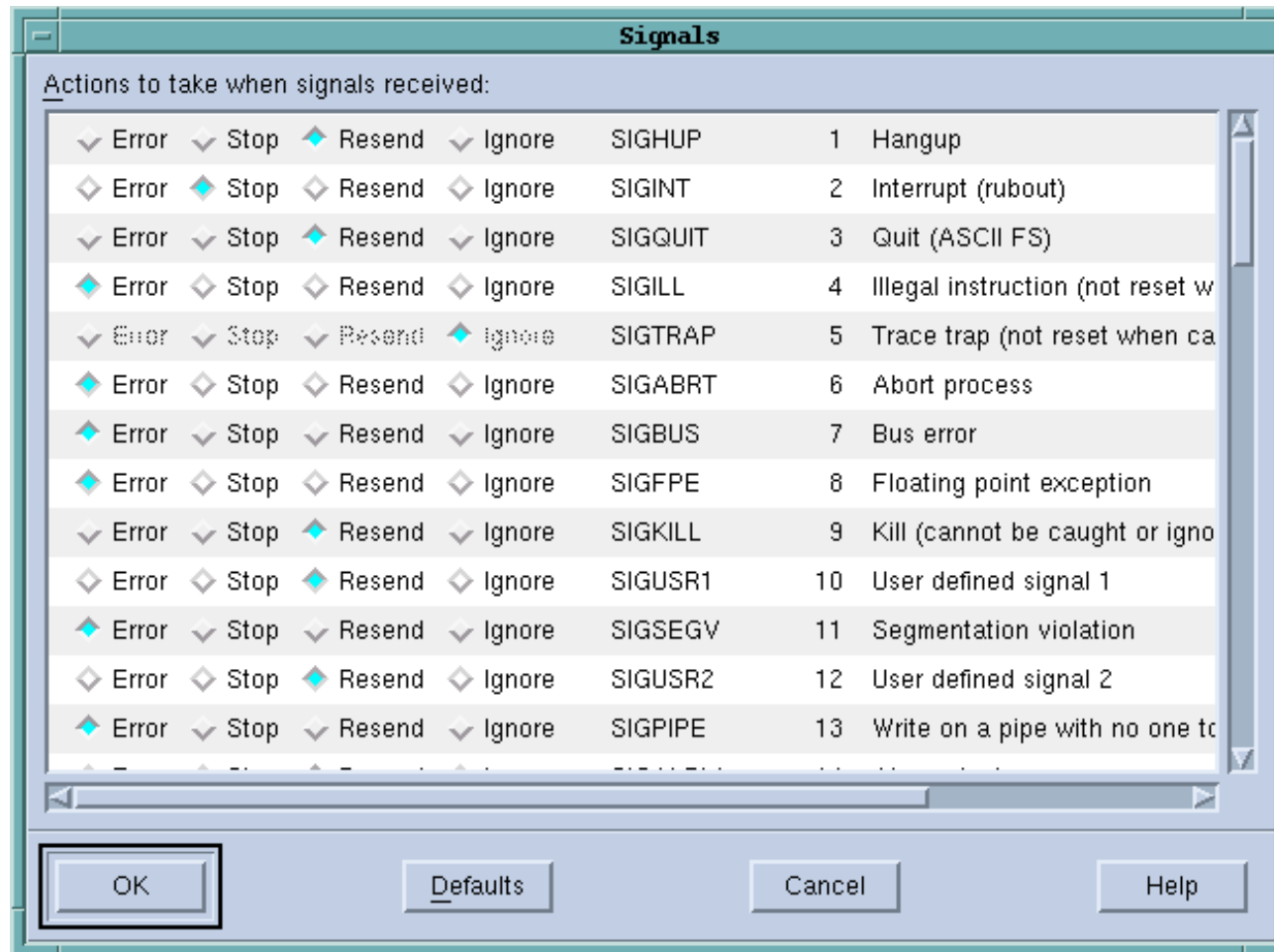
Uses Hardware Watchpoints with various limitations based on architecture

Using Set PC to resume execution at an arbitrary point



Managing Signals

File > Signals



Error

Stop

Resend

Ignore

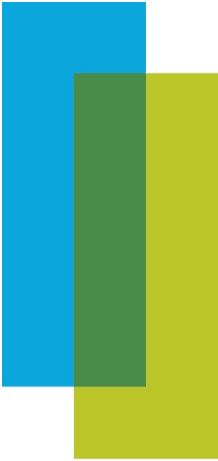
Stop the process and flag as error

Stop the process

Pass the signal to the target and do nothing: use with signal handlers

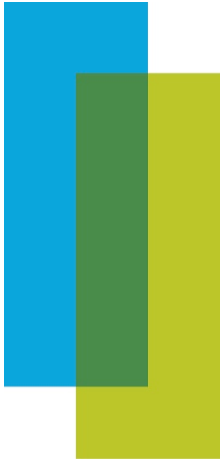
Discard the signal





TotalView Basics

Viewing and Editing Data



Diving on Variables

You can use Diving to:

- ... get more information
- ... open a variable in a Variable Window.
- ... chase pointers in complex data structures
- ... refocus the Process window Source Pane

You can Dive on:

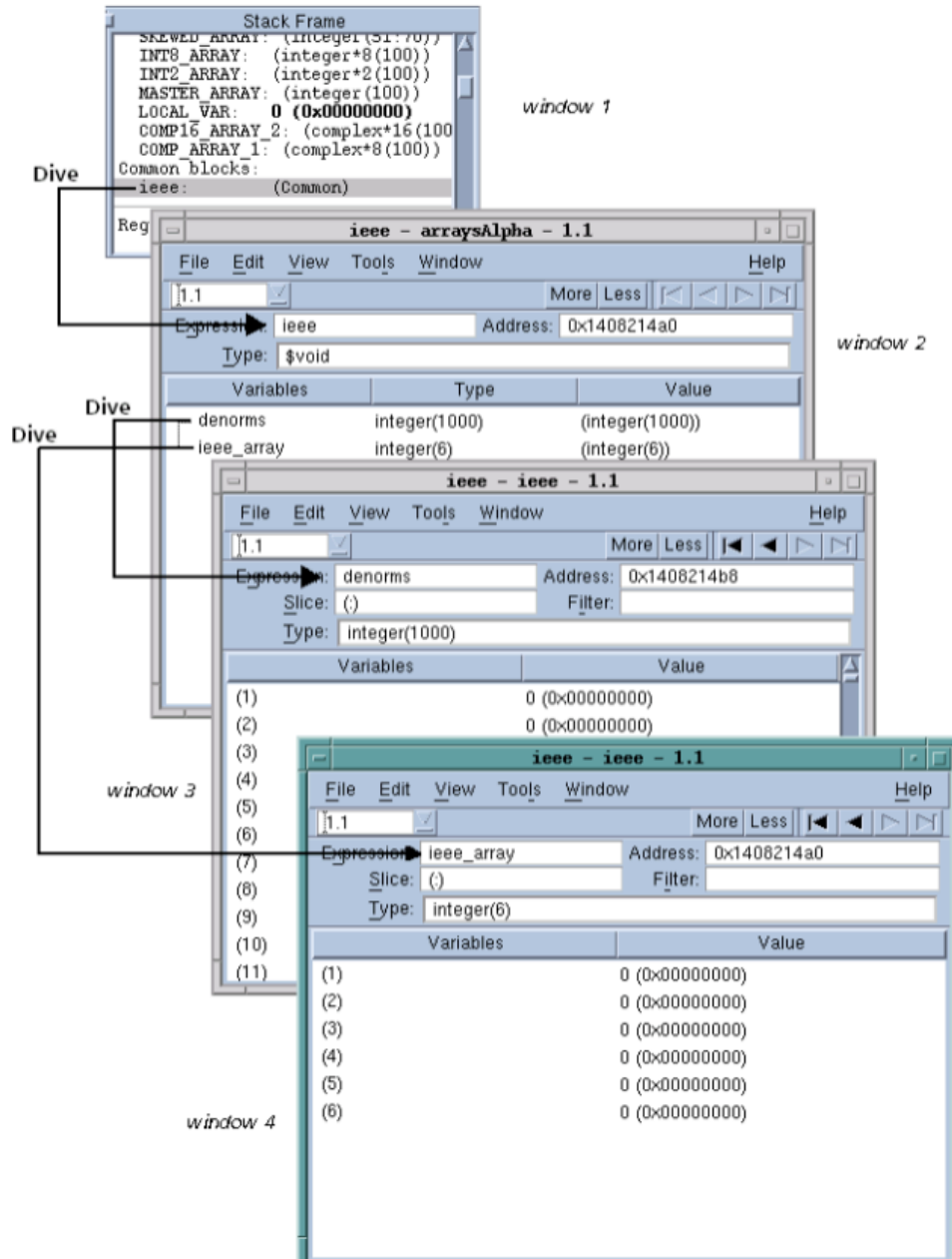
- ... variable names to open a variable window
- ... function names to open the source in the Process Window.
- ... processes and threads in the Root Window.

How do I dive?

- Double-click the left mouse button on selection
- Single-click the middle mouse button on selection.
- Select Dive from context menu opened with the right mouse button

Diving

Diving on a Common Block in the Stack Frame Pane



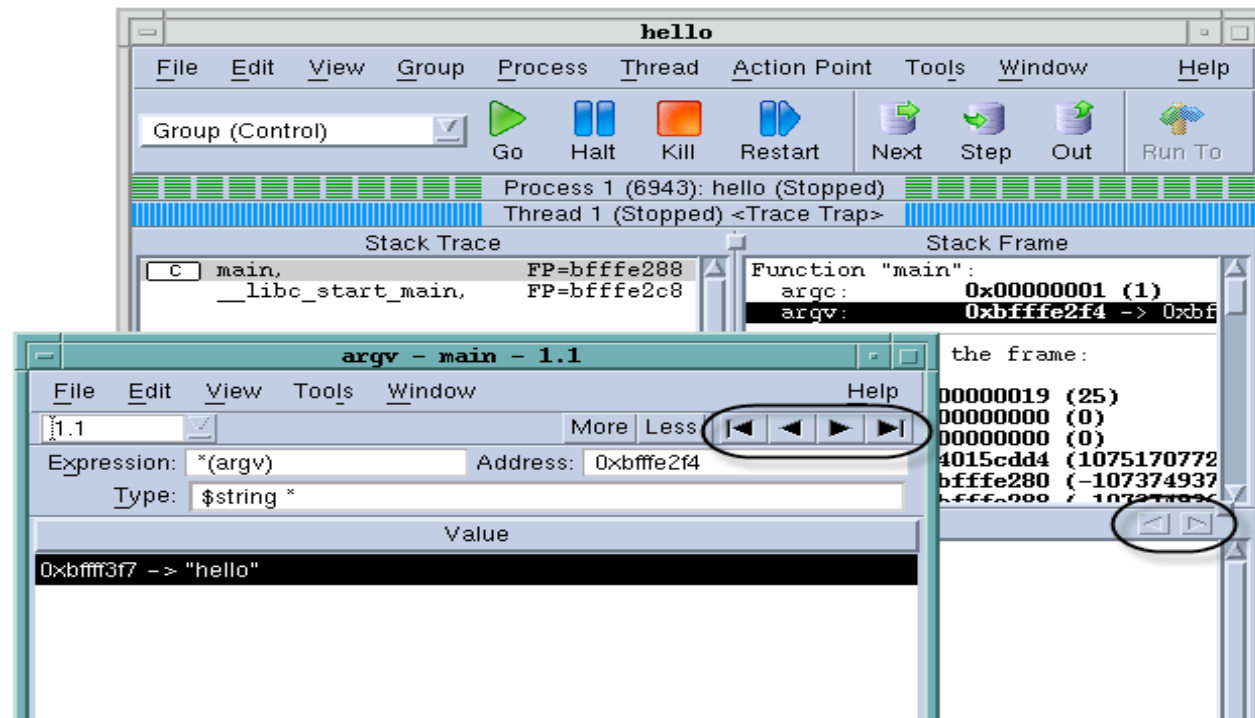
The diagram illustrates the process of diving into a common block in the stack frame pane across four windows:

- window 1:** Shows the Stack Frame pane with a list of variables. The 'Common blocks' section is expanded, showing 'ieee: (Common)'. A 'Dive' arrow points from this entry to window 2.
- window 2:** Shows the 'ieee - arraysAlpha - 1.1' window. The 'Expression' field contains 'ieee' and the 'Type' is '\$void'. A 'Dive' arrow points from the 'Variables' table to window 3.
- window 3:** Shows the 'ieee - ieee - 1.1' window. The 'Expression' field contains 'denorms' and the 'Type' is 'Integer(1000)'. A 'Dive' arrow points from the 'Variables' table to window 4.
- window 4:** Shows the 'ieee - ieee - 1.1' window. The 'Expression' field contains 'ieee_array' and the 'Type' is 'Integer(6)'. The 'Variables' table shows a list of values, all of which are '0 (0x00000000)'.

| Variables | Type | Value |
|------------|---------------|-----------------|
| denorms | integer(1000) | (integer(1000)) |
| ieee_array | integer(6) | (integer(6)) |

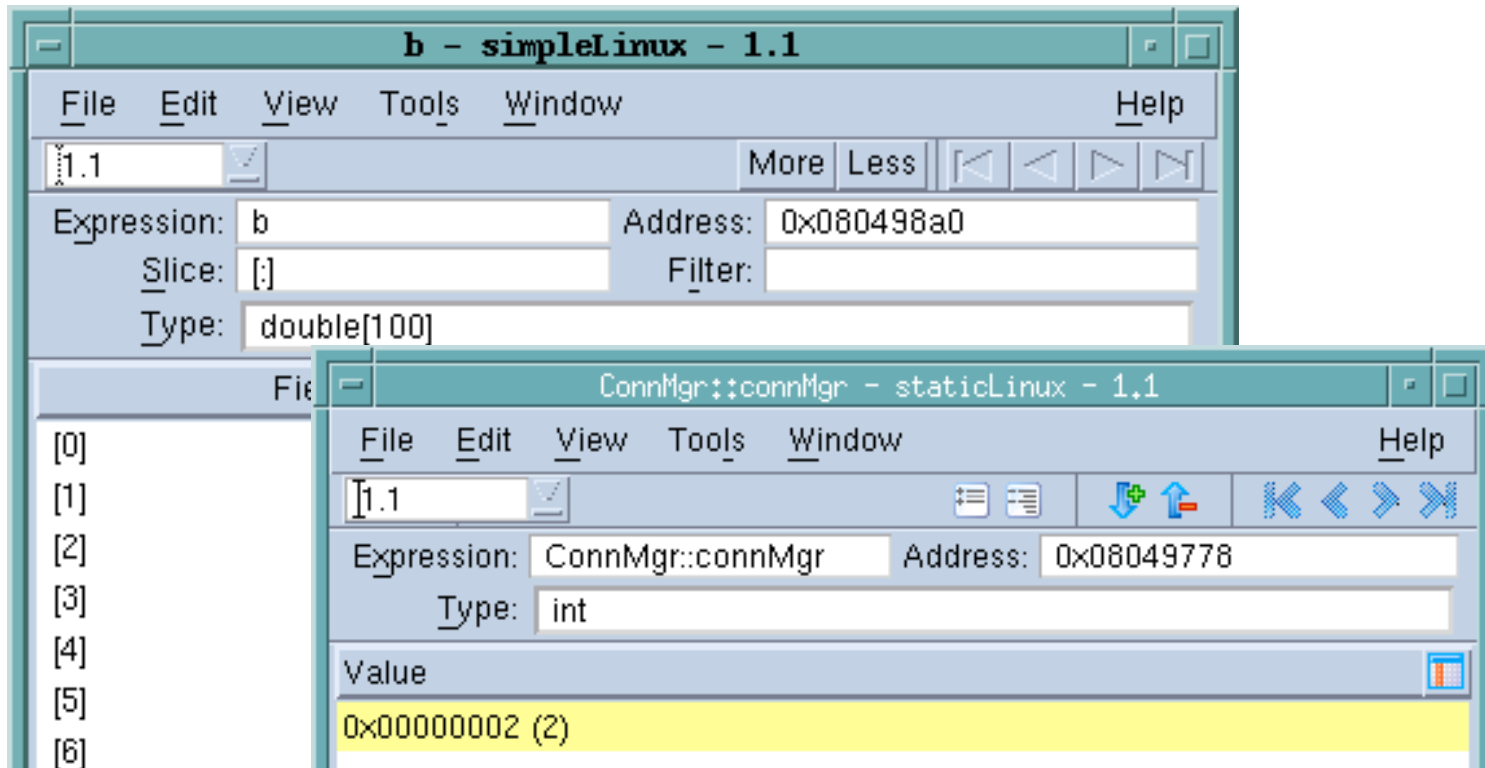
| Variables | Value |
|-----------|----------------|
| (1) | 0 (0x00000000) |
| (2) | 0 (0x00000000) |
| (3) | 0 (0x00000000) |
| (4) | 0 (0x00000000) |
| (5) | 0 (0x00000000) |
| (6) | 0 (0x00000000) |

Undiving



In a Process Window: retrace the path that has been explored with multiple dives.
 In a Variable Window: replace contents with the previous contents.

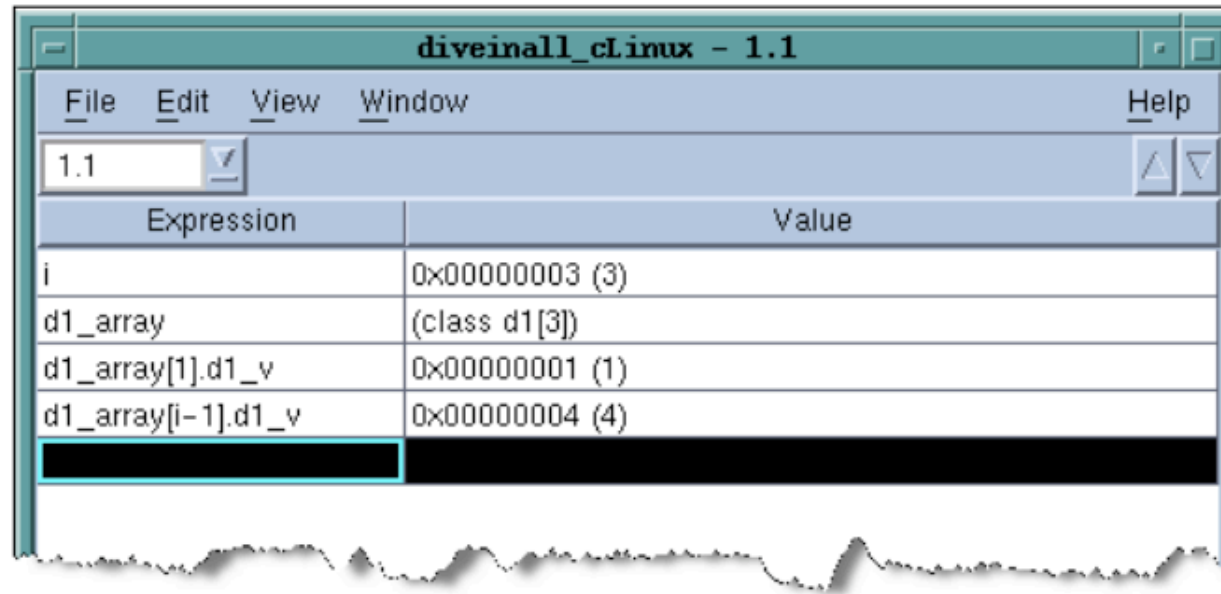
The Variable Window



Editing Variables

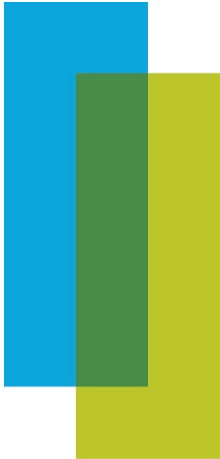
- Window contents are updated automatically
- Changed values are highlighted
- “Last Value” column is available
- You can remove changes with Edit > Reset Default
- Click once on the value
- Cursor switches into edit mode
- Esc key cancels editing
- Enter key commits a change
- Editing values changes the memory of the program

Expression List Window



Add to the expression list using contextual menu with right-click on a variable, or by typing an expression directly in the window

- Reorder, delete, add
- Sort the expressions
- Edit expressions in place
- Dive to get more info
- Updated automatically
- Expression-based
- Simple values/expressions
- View just the values you want to monitor



Viewing Arrays

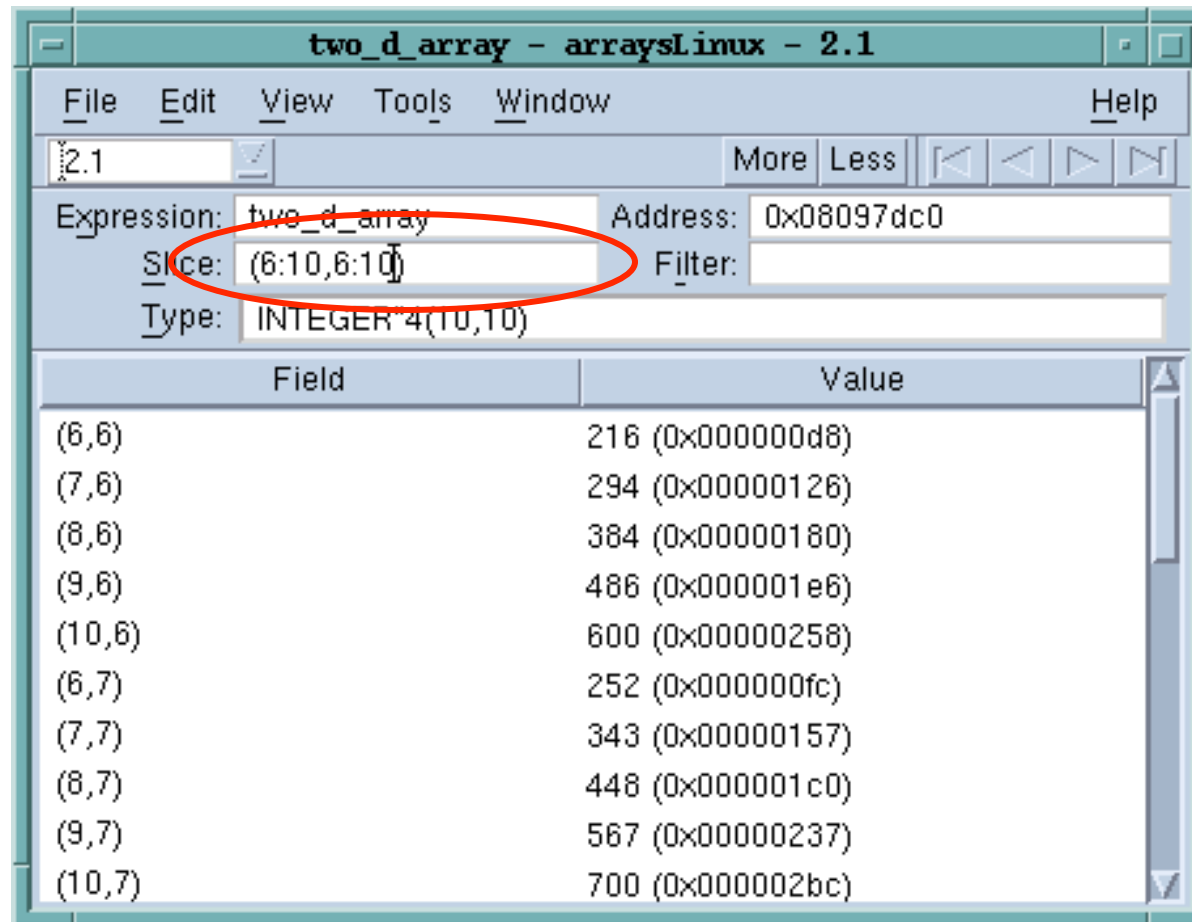
Data Arrays

| AN_ARRAY - ten_by_tenAlpha - 1.1 | |
|----------------------------------|----------------------|
| File Edit View Tools Window Help | |
| 1.1 | More Less |
| Expression: AN_ARRAY | Address: 0x1400011c0 |
| Slice: (:::) | Filter: |
| Type: real(10,10,10) | |
| Field | Value |
| (1,1,1) | 0 |
| (2,1,1) | -0.506366 |
| (3,1,1) | -0.873297 |
| (4,1,1) | -0.999756 |
| (5,1,1) | |
| (6,1,1) | |
| (7,1,1) | |
| (8,1,1) | |
| (9,1,1) | |
| (10,1,1) | |

| d1_array - main - 1.1 | | |
|----------------------------------|---------------------|----------------------------|
| File Edit View Tools Window Help | | |
| 1.1 | More Less | |
| Expression: d1_array | Address: 0xbfffeb30 | |
| Slice: [] | Filter: | |
| Type: class d1[3] | | |
| Field | Type | Value |
| [0] | class d1 | (Class) |
| base | class base | (Virtual public base class |
| b_v | int | 0x00000000 (0) |
| bb_v | int | 0x00000000 (0) |
| name | \$string * | 0x08048ad9 -> "base" |
| base2 | class base2 | (Virtual public base class |
| b2_v | int | 0x00000000 (0) |
| bb_v2 | int | 0x00000000 (0) |
| name | \$string * | 0x08048ade -> "base2" |
| d1_v | int | 0x00000000 (0) |

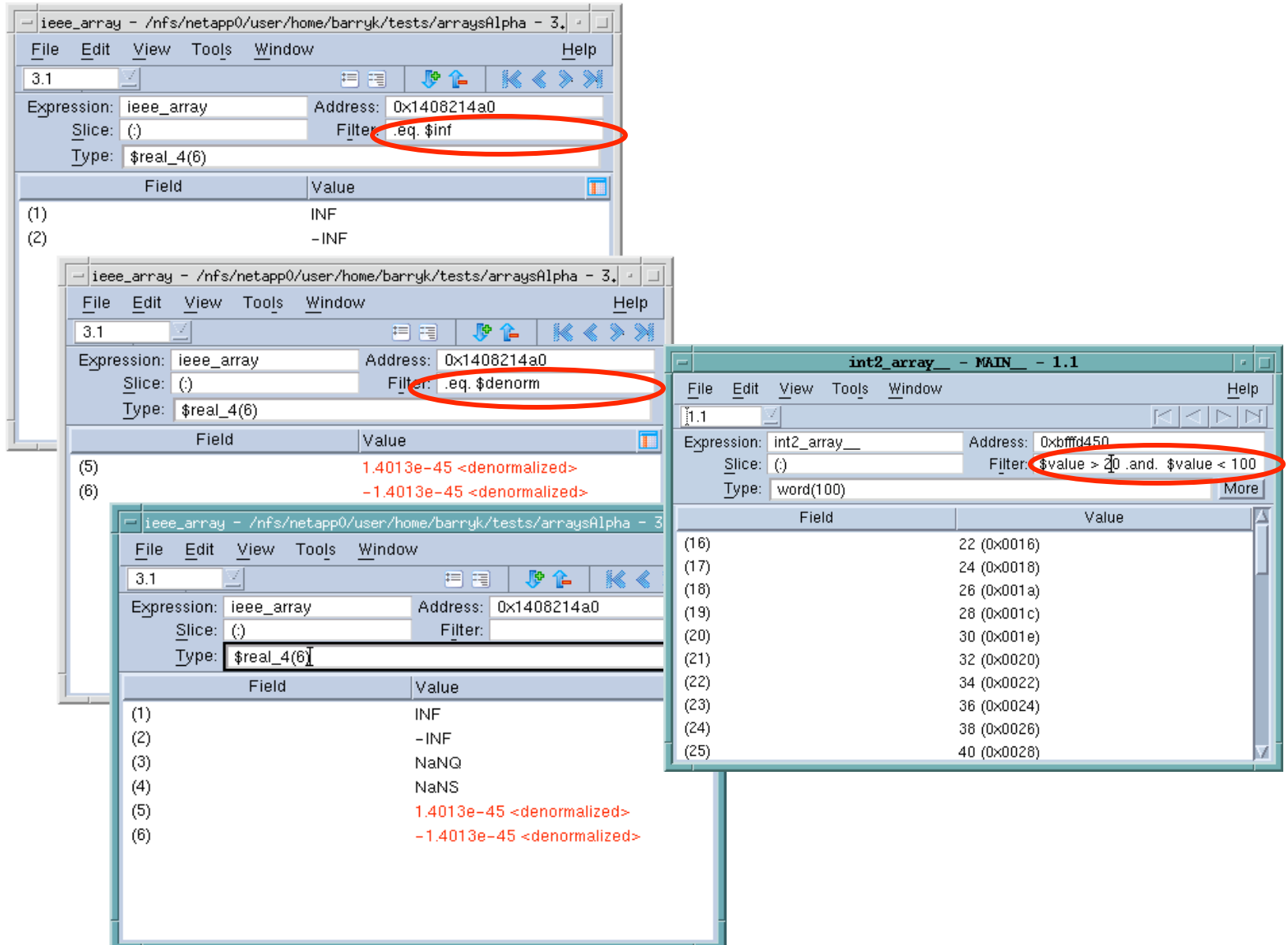
Structure Arrays

Slicing Arrays



Slice notation is `[start:end:stride]`

Filtering Arrays



Screenshot 1: Filtering for infinity values

Expression: `ieee_array` Address: `0x1408214a0`
Slice: `()` Filter: `.eq. $inf`
Type: `$real_4(6)`

| Field | Value |
|-------|-------|
| (1) | INF |
| (2) | -INF |

Screenshot 2: Filtering for denormalized values

Expression: `ieee_array` Address: `0x1408214a0`
Slice: `()` Filter: `.eq. $denorm`
Type: `$real_4(6)`

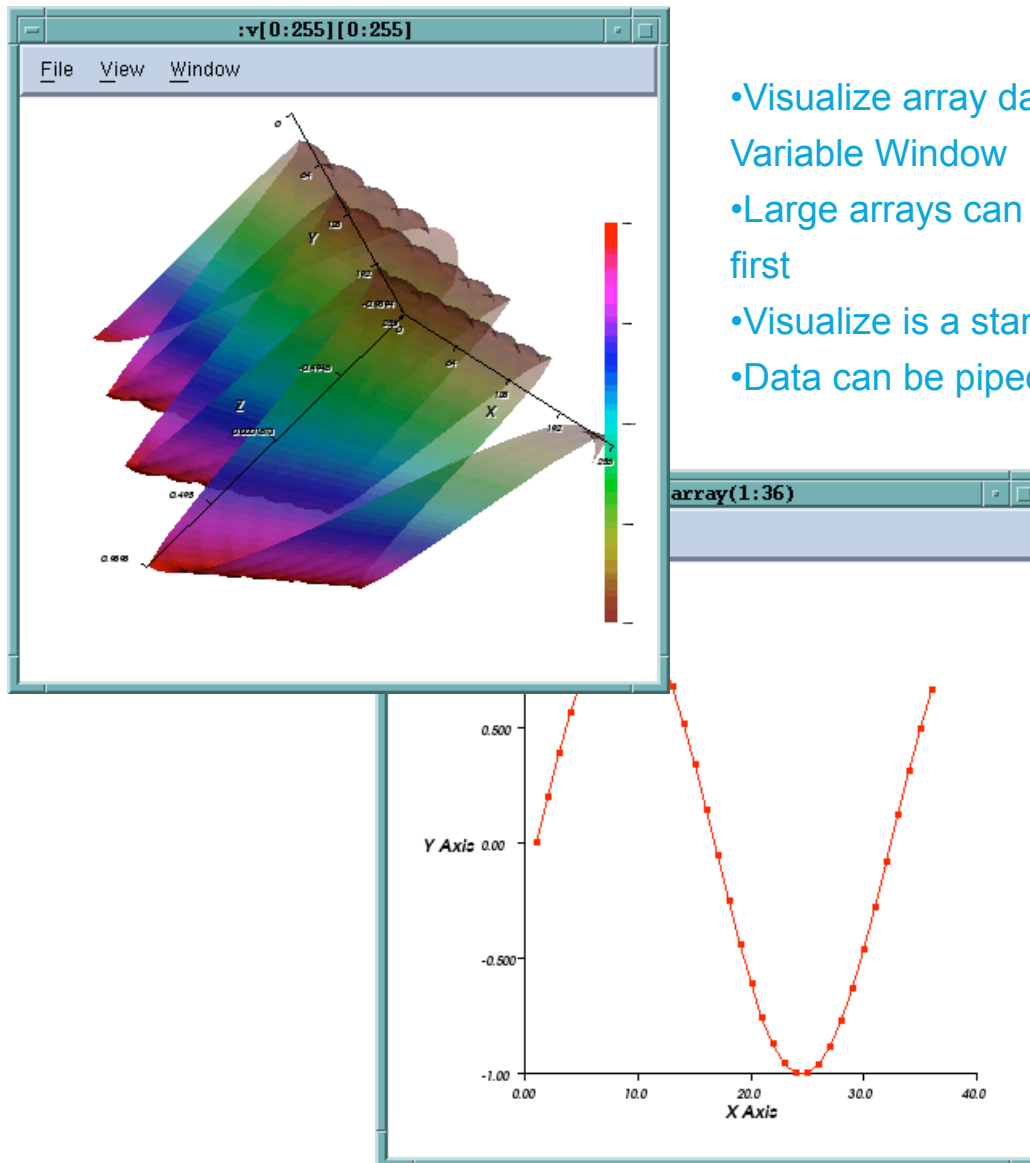
| Field | Value |
|-------|----------------------------|
| (5) | 1.4013e-45 <denormalized> |
| (6) | -1.4013e-45 <denormalized> |

Screenshot 3: Filtering for values between 20 and 100

Expression: `int2_array__` Address: `0xbfffd450`
Slice: `()` Filter: `$value > 20 .and. $value < 100`
Type: `word(100)`

| Field | Value |
|-------|-------------|
| (16) | 22 (0x0016) |
| (17) | 24 (0x0018) |
| (18) | 26 (0x001a) |
| (19) | 28 (0x001c) |
| (20) | 30 (0x001e) |
| (21) | 32 (0x0020) |
| (22) | 34 (0x0022) |
| (23) | 36 (0x0024) |
| (24) | 38 (0x0026) |
| (25) | 40 (0x0028) |

Visualizing Arrays



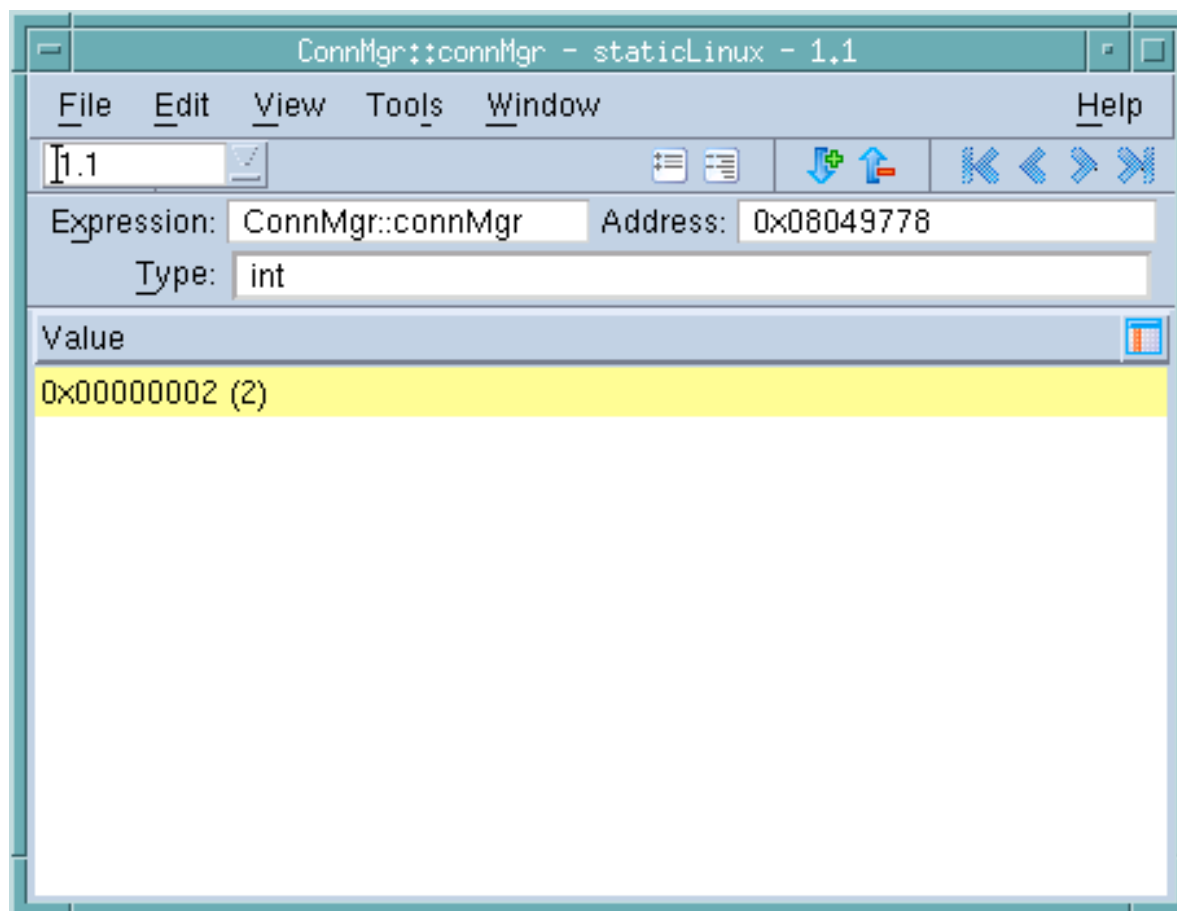
- Visualize array data using Tools > Visualize from the Variable Window
- Large arrays can be sliced down to a reasonable size first
- Visualize is a standalone program
- Data can be piped out to other visualization tools

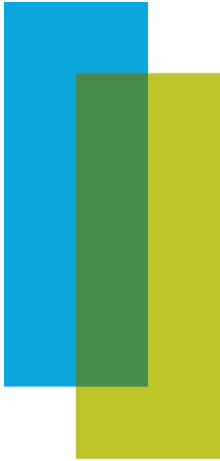
- Visualize allows to spin, zoom, etc.
- Data is not updated with Variable Window; You must revisualize
- \$visualize() is a directive in the expression system, and can be used in evaluation point expressions.

Four Ways to Look at Variables

- **Glance**
 - Stack frame
- **Hover**
 - Source pane
- **Dive to data window**
 - Source, Stack or Variable Window
 - Arrays, structures, explore
- **Monitor via expression list**
 - Source, Stack or Variable window
 - Keep an eye on scalars and expressions

Variable window fields are editable



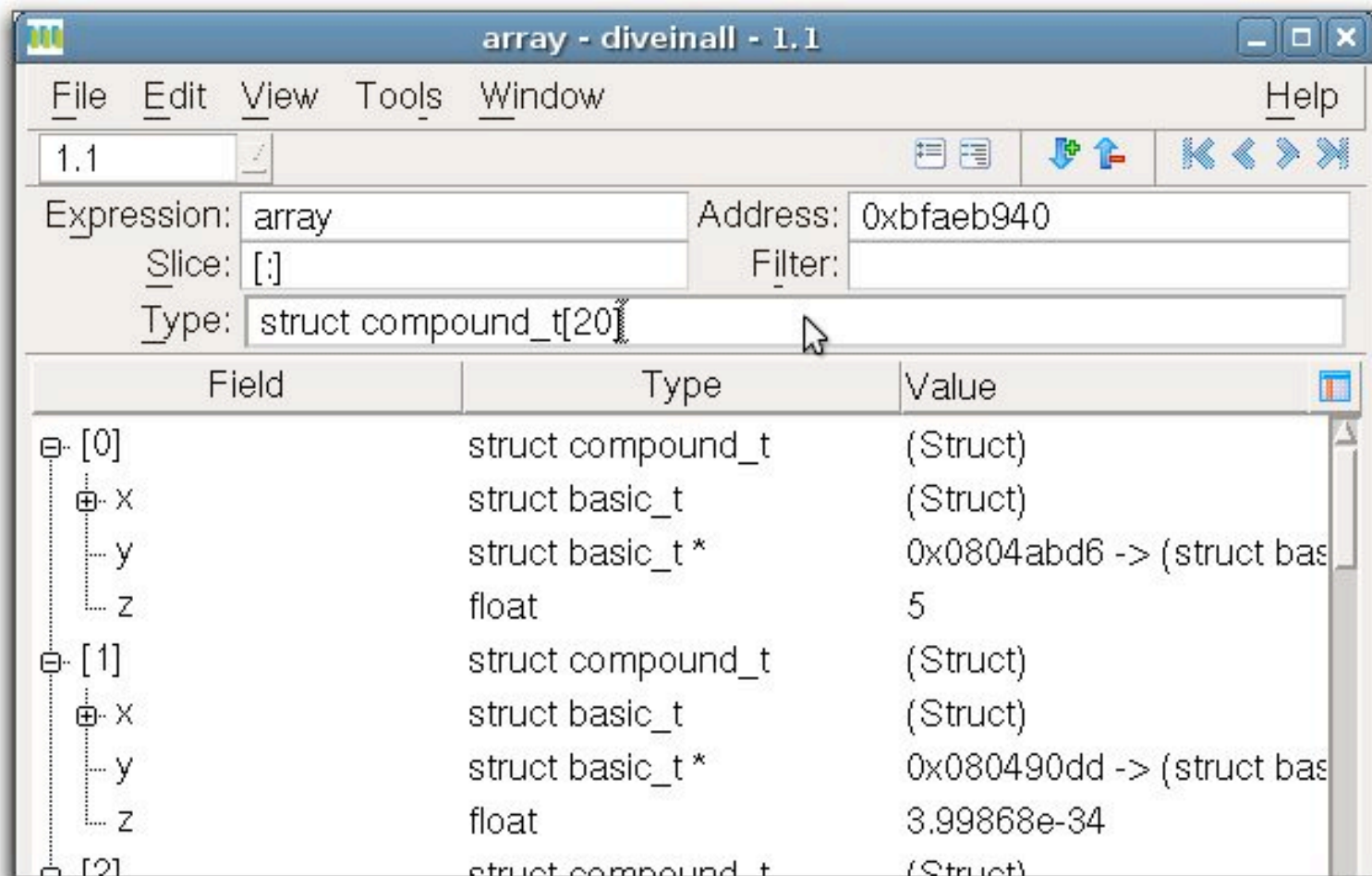


Typecasting Variables

- Edit the type of a variable
- Changes the way TotalView interprets the data in your program
- Does not change the data in your program
- Often used with pointers
- Type cast to a void or code type to snoop around in memory

Give TotalView a starting memory address and TotalView will interpret and display your memory from that location.

Typcasting a Dynamic Array



Dive in All

The image shows three overlapping screenshots of the TotalView debugger interface, demonstrating the 'Dive in All' feature.

Left Window: Shows a memory dump of an array of `struct compound_t` elements. The expression is `array` at address `0xbfaeb940`. The type is `struct compound_t[20]`. The dump shows elements `[0]` through `[3]`, each containing fields `x`, `a`, `b`, `c`, `y`, and `z` with various data types (struct, int, float).

Middle Window: Shows a context menu with the following options: `Dive`, `Dive in New Window`, `Dive in All` (highlighted), `Add to Expression List`, `Create Watchpoint`, and `Change Field`.

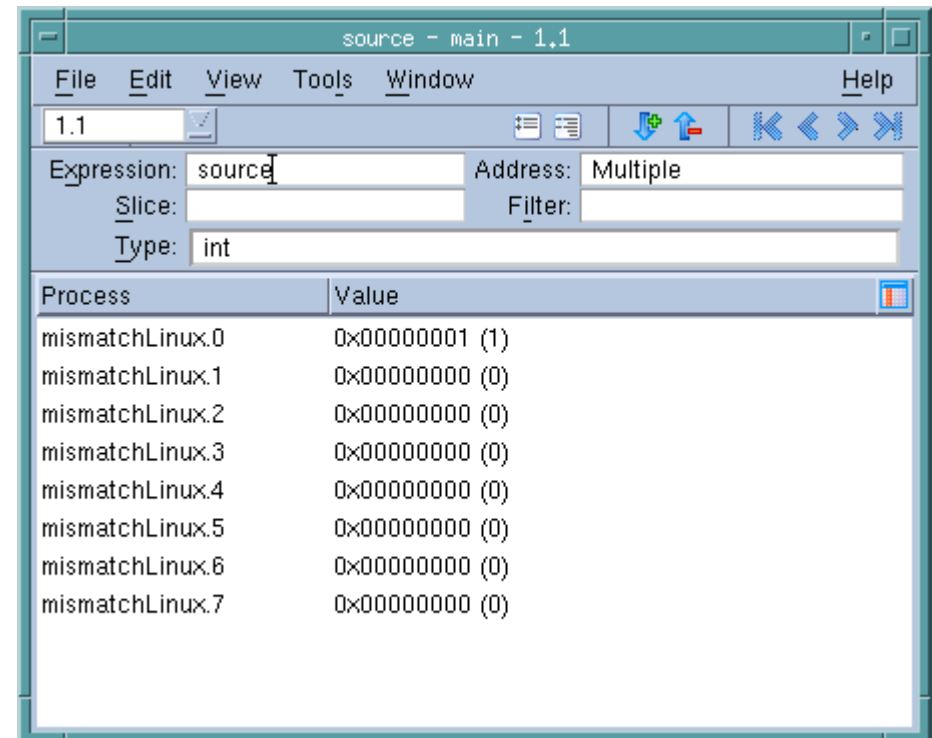
Right Window: Shows the result of the 'Dive in All' operation. The expression is `((struct compound_t[20])arr)` at address `0xbfaeb940` [Sparse]. The type is `int[20]`. The dump shows a list of 20 integer values, each corresponding to an element in the original array.

| Field | Value |
|-------|--------------------------|
| [0] | 0xb7eb3adc (-1209320740) |
| [1] | 0xbfaeb968 (-1079068312) |
| [2] | 0x08048ddd (134516189) |
| [3] | 0xb7fe7778 (-1208027272) |
| [4] | 0xbfaeb9b0 (-1079068240) |
| [5] | 0xbfaeba2c (-1079068116) |
| [6] | 0xbfaeb9f8 (-1079068168) |
| [7] | 0x00000000 (0) |
| [8] | 0xb7fe4b30 (-1208071376) |
| [9] | 0x00000000 (0) |
| [10] | 0x0804d004 (134533124) |
| [11] | 0x00000001 (1) |
| [12] | 0xbfaed811 (-1079060463) |
| [13] | 0xbfaed8c0 (-1079060288) |
| [14] | 0xbfaedbd2 (-1079059502) |
| [15] | 0xbfaedc90 (-1079059312) |
| [16] | 0xbfaedddd (-1079058979) |
| [17] | 0xbfaede92 (-1079058798) |
| [18] | 0xbfaedf3e (-1079058626) |
| [19] | 0xbfaedfbc (-1079058500) |

Dive in All will display
an element in an array
of structures as if
it were a simple array.

Looking at Variables across Processes

- **TotalView allows you to look at the value of a variable in all MPI processes**
 - Right Click on the variable
 - Select the View > View Across
- **TotalView creates an array indexed by process**
- **You can filter and visualize**
- **Use for viewing distributed arrays as well.**

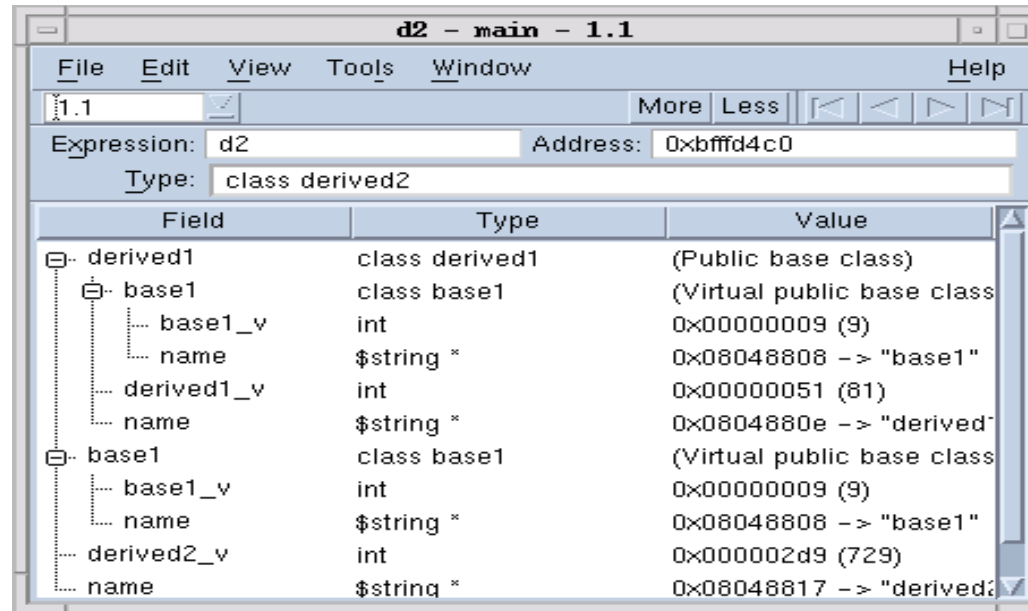


The screenshot shows the TotalView interface with the 'View Across' window open. The window title is 'source - main - 1.1'. The menu bar includes File, Edit, View, Tools, Window, and Help. The toolbar contains icons for file operations and navigation. The 'Expression' field is set to 'source', 'Address' is 'Multiple', 'Slice' is empty, 'Filter' is empty, and 'Type' is 'int'. Below this is a table showing the values of 'source' across different MPI processes.

| Process | Value |
|-----------------|----------------|
| mismatchLinux.0 | 0x00000001 (1) |
| mismatchLinux.1 | 0x00000000 (0) |
| mismatchLinux.2 | 0x00000000 (0) |
| mismatchLinux.3 | 0x00000000 (0) |
| mismatchLinux.4 | 0x00000000 (0) |
| mismatchLinux.5 | 0x00000000 (0) |
| mismatchLinux.6 | 0x00000000 (0) |
| mismatchLinux.7 | 0x00000000 (0) |

C++ Class Hierarchies

Variable Window shows class hierarchy using indentation



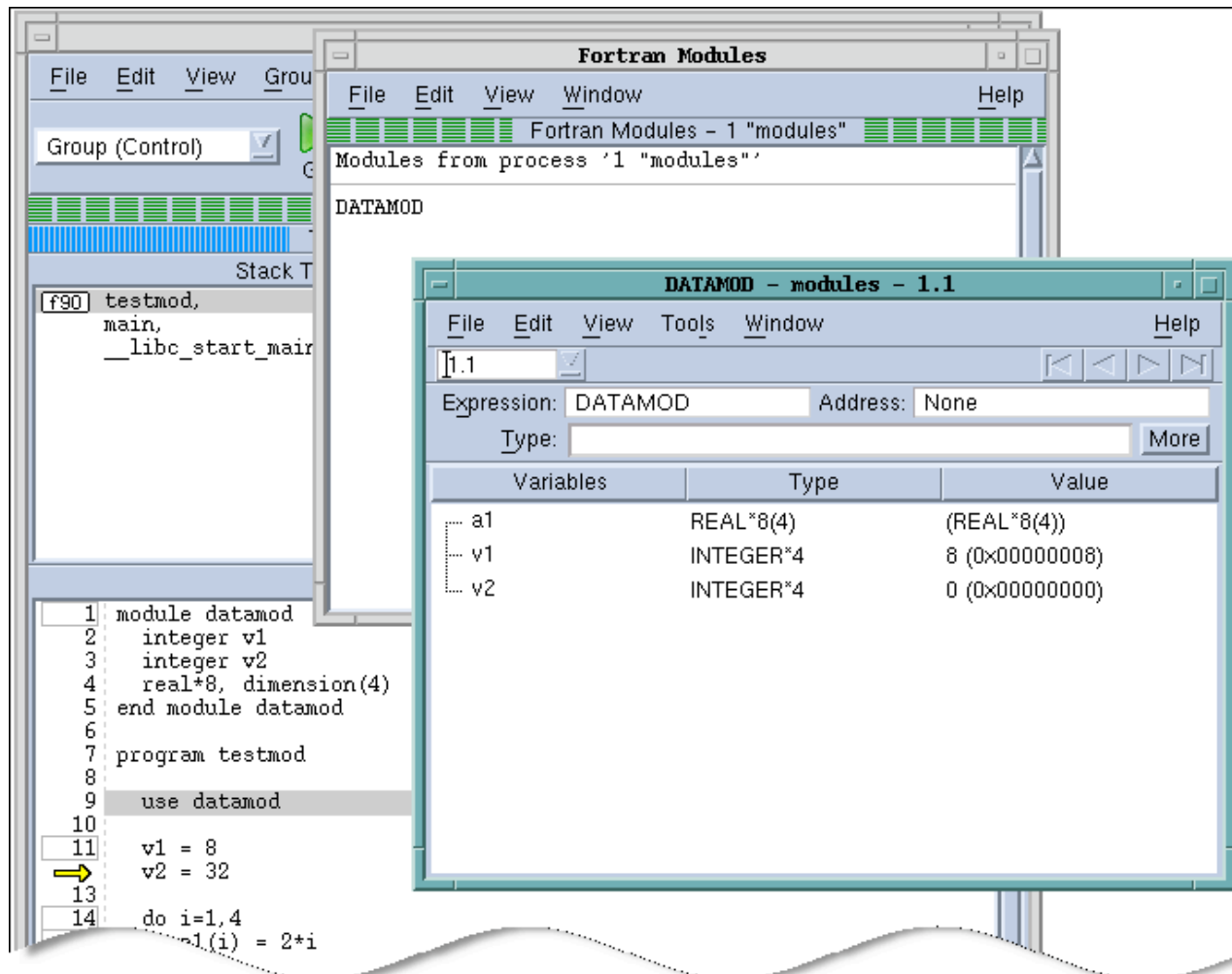
- Example:**
- derived2 inherits from base1 and derived1
 - derived1 inherits from base1

Note:

- Virtual public base classes appear each time they are referenced
- The vtable entry here is part of the C++ implementation but can provide useful information

Fortran 90 Modules

Tools > Fortran Modules



The screenshot displays the TotalView Fortran Modules tool interface. The main window, titled "Fortran Modules", shows a list of modules from process '1 "modules"', with "DATAMOD" selected. A sub-window, titled "DATAMOD - modules - 1.1", provides details for the selected module. It includes a menu bar (File, Edit, View, Tools, Window, Help), a version field set to "1.1", and input fields for "Expression: DATAMOD" and "Address: None". Below these is a "More" button. A table lists the module's variables:

| Variables | Type | Value |
|-----------|-----------|----------------|
| a1 | REAL*8(4) | (REAL*8(4)) |
| v1 | INTEGER*4 | 8 (0x00000008) |
| v2 | INTEGER*4 | 0 (0x00000000) |

The background shows a Fortran 90 source code editor with the following code:

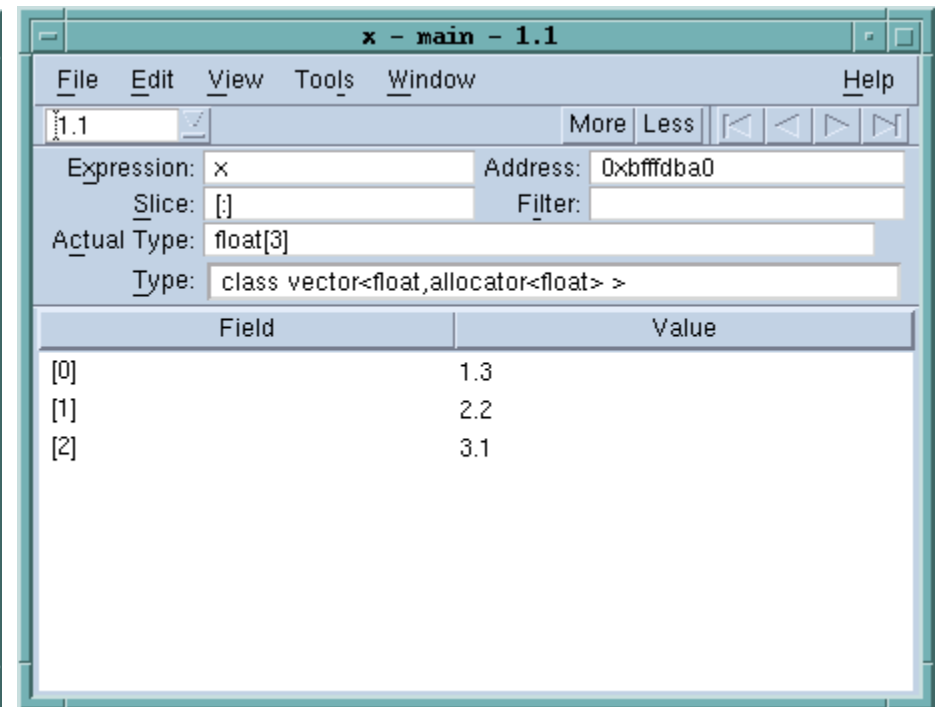
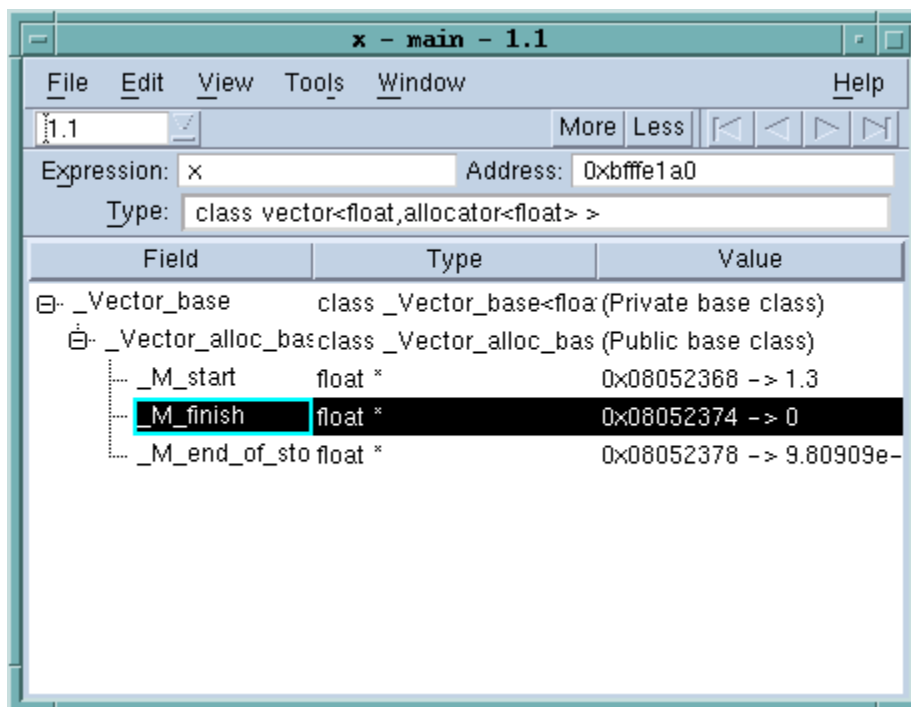
```
1 module datamod
2   integer v1
3   integer v2
4   real*8, dimension(4)
5 end module datamod
6
7 program testmod
8
9   use datamod
10
11   v1 = 8
12   v2 = 32
13
14   do i=1,4
15     v1(i) = 2*i
```

STLView

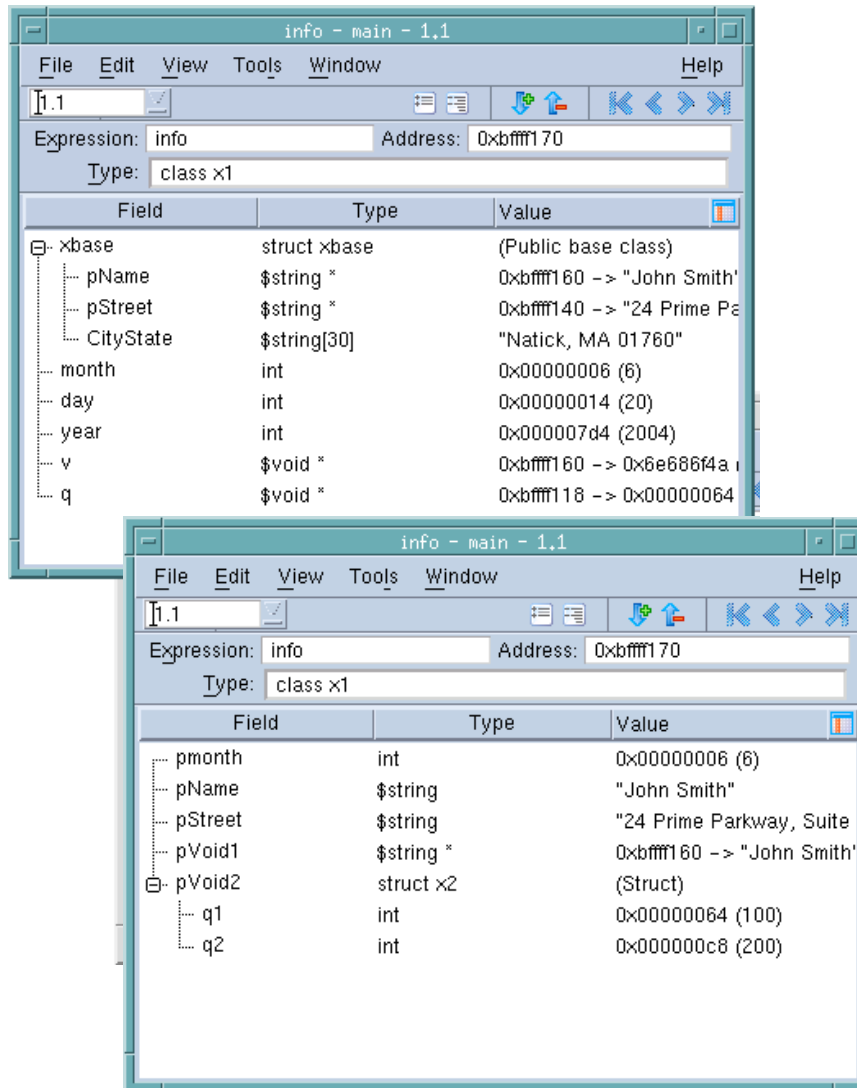


STLView transforms templates into readable and understandable information

- STLView supports `std::vector`, `std::list`, `std::map`, `std::string`
- See documentation for which STL implementations are supported



Creating Type Transformations



In \$HOME/.tvdrc:

```
::TV::TTF::RTF::build_struct_transform {
  name {^class x1$}
  members {
    { pmonth { month } }
    { pName { xbase upcast { * pName } } }
    { pStreet { xbase upcast { * pStreet } } }
    { pVoid1 { "$string *" cast v } }
    { pVoid2 { * { "class x2 *" cast q } } }
  }
}
```

Meta Language:

{member}

{* expr}

{expr . Expr}

{expr -> expr}

{datatype case expr}

{baseclass upcast expr}



Documentation

www.totalviewtech.com



Software Tools for
Your Parallel Debugging Needs



[CONTACT US](#)

[RESELLERS](#)

[CUSTOMER PORTAL](#)

[STORE](#)

[OUR SOFTWARE](#)

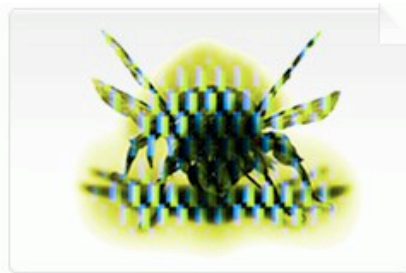
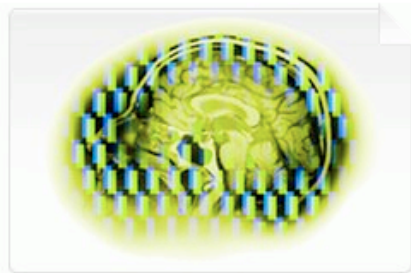
[OUR CUSTOMERS](#)

[DOCUMENTATION &
TRAINING](#)

[USER COMMUNITY](#)

[DOWNLOADS &
SUPPORT](#)

[ABOUT US](#)



Participate in the latest
Totalview, MemoryScape & ReplayEngine
Betas

Rogue Wave
has acquired
TotalView Technologies, Inc.

IN THE NEWS

01.06.10
Rogue Wave Software has acquired
TotalView Technologies, Inc.

RESOURCES

- Video Tutorials
- Case Studies
- White Papers
- User Forum
- Platform Support

OUR SOFTWARE

TotalView MemoryScape
ReplayEngine Workbench
Remote Display Student Express

LEARN MORE

Using TotalView to Validate C++ Translations of MATLAB Models

TotalView is developers' key tool in comparing different versions of an application under development and in testing... > [more](#)

MPI Debugging Standards ... What Took So Long?!

It has been said, "The nice thing about standards is that there are so many of them to choose from." But when it comes to standard debugging interfaces for parallel computers... > [more](#)

Manage the risk in computationally intensive parallel programs

With support for 20 implementations of MPI, as well as OpenMP and UPC, we help you manage the risk of parallel applications... > [more](#)

WHAT'S NEW

➔ **Rogue Wave has
acquired TotalView
Technologies, Inc.**
- [FAQs](#)

➔ **Announcing NVIDIA GPU
Port of TotalView**

➔ **TotalView Supports Cray
CX1 Deskside
Supercomputer**

➔ **InsideHPC Interview
TotalView on Extreme
Scale Development**

➔ **Troubleshooting Network
Applications White Paper**


 **Download
Free Trial**

MATLAB AND TOTALVIEW



White Paper:
Validating C++
Translations of
MATLAB Models with
TotalView

Using TotalView to
create cutting-edge
applications with highly
specialized computational
challenges.

See the [video](#) 

Support

[Home](#) | [Support](#) | [TotalView Documentation](#)

- [New Features](#)
- [Full User Guide](#)
- [Reference Guide](#)
- [GUI Reference Guide](#)
- [Platforms and System Requirements](#)
- [Installing TotalView](#)
- [Getting Started](#)
- [Using Threads Processes and Threads](#)
- [Using the Remote Display Client](#)
- [Setting Up a Debugging Session](#)
- [Setting Up MPI Debugging Sessions](#)
- [Setting Up Parallel Debugging Sessions](#)
- [Setting Up tvdsvr](#)
- [Command Line Syntax](#)
- [Using TotalView Windows](#)
- [Visualizing Programs and Data](#)
- [Debugging Programs](#)
- [Examining and Changing Data](#)
- [Examining Arrays](#)
- [Setting Action Points](#)
- [Evaluating Expressions](#)
- [Debugging Memory Problems](#)
- [Using the CLI](#)
- [Creating Type Transformations](#)



**Download
Free Trial**

NEXT STEPS



[Contact Sales](#)



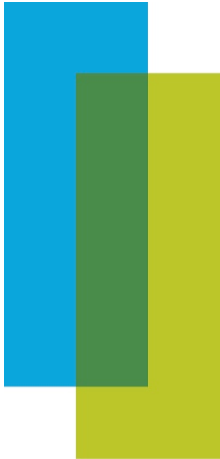
[View License Options](#)



[Request Web Demo](#)



[Sign up for The View](#)



Developers & Support

[Home](#) | [Developers & Support](#) | [Video Tutorials](#)

Video Tutorials for TotalView, MemoryScape and ReplayEngine

These on demand video tutorials provide you with an easy and efficient way to learn the key capabilities of our products. Whether you are currently evaluating TotalView, ReplayEngine or MemoryScape, or are looking for a brief update on the latest versions, these short videos are an ideal self paced learning tool.

» [C/C++ Support](#)

6 MINUTES

» [Deterministic Replay with ReplayEngine](#)

5 MINUTES

» [Getting Started with TotalView](#)

14 MINUTES

» [Memory Debugging](#)

5 MINUTES

» [Asynchronous Threads](#)

2 MINUTES

» [Threads Navigation](#)

2.5 MINUTES

» [Setting Breakpoint Preferences for Threads](#)

2.5 MINUTES

» [Viewing Data Across Threads](#)

2.5 MINUTES

» [Debugging OpenMP with TotalView](#)

2.5 MINUTES


» [Seeing Threads and Process Status](#)

2 MINUTES


On Demand Webinars


» [TotalView Technologies ReplayEngine demo](#)


» [TotalView Technologies TotalView demo](#)


 **Download Free Trial**

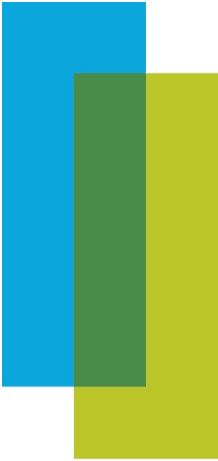
NEXT STEPS

 [Contact Sales](#)

 [View License Options](#)

 [Request Web Demo](#)

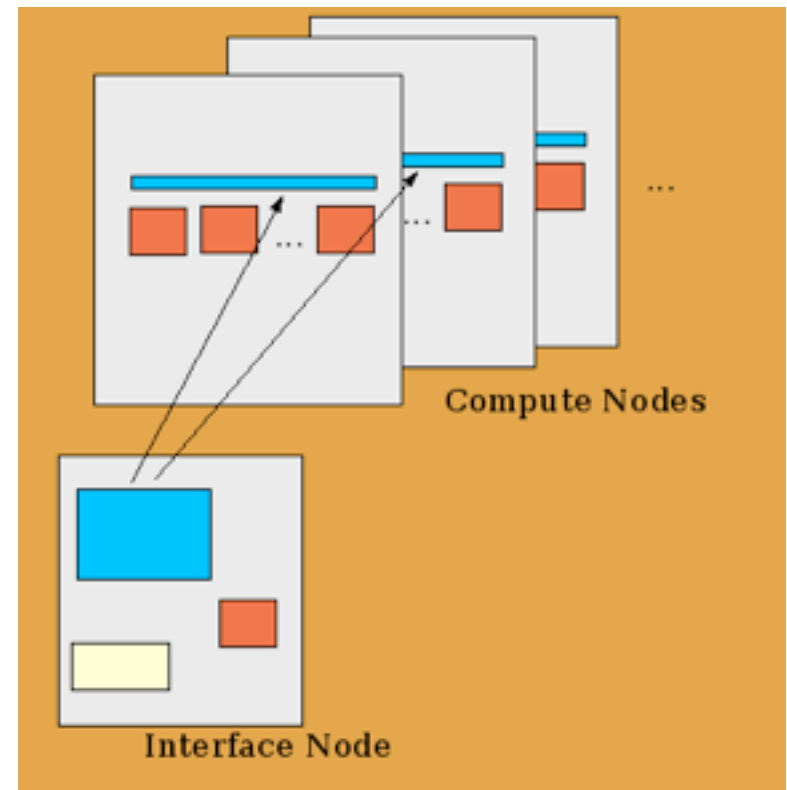
 [Sign up for The View](#)



Parallel Debugging

Architecture for Cluster Debugging

- **Single Front End (TotalView)**
 - GUI
 - debug engine
- **Debugger Agents (tvdsvr)**
 - Low overhead, 1 per node
 - Traces multiple rank processes
- **TotalView communicates directly with tvdsvrs**
 - Not using MPI
 - Protocol optimization



Provides Robust, Scalable and efficient operation with Minimal Program Impact

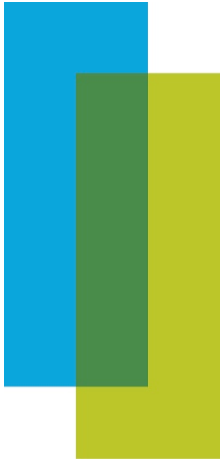
Debugging Multithreaded Programs

When debugging multithreaded programs, you want to:

- Know where to look to get thread status.
- Be able to switch the focus from one thread to another quickly and easily.
- Understand how asynchronous thread control commands (step, go, halt) and breakpoints are used.

A parallel program has a lot more states than 'Running or stopped'

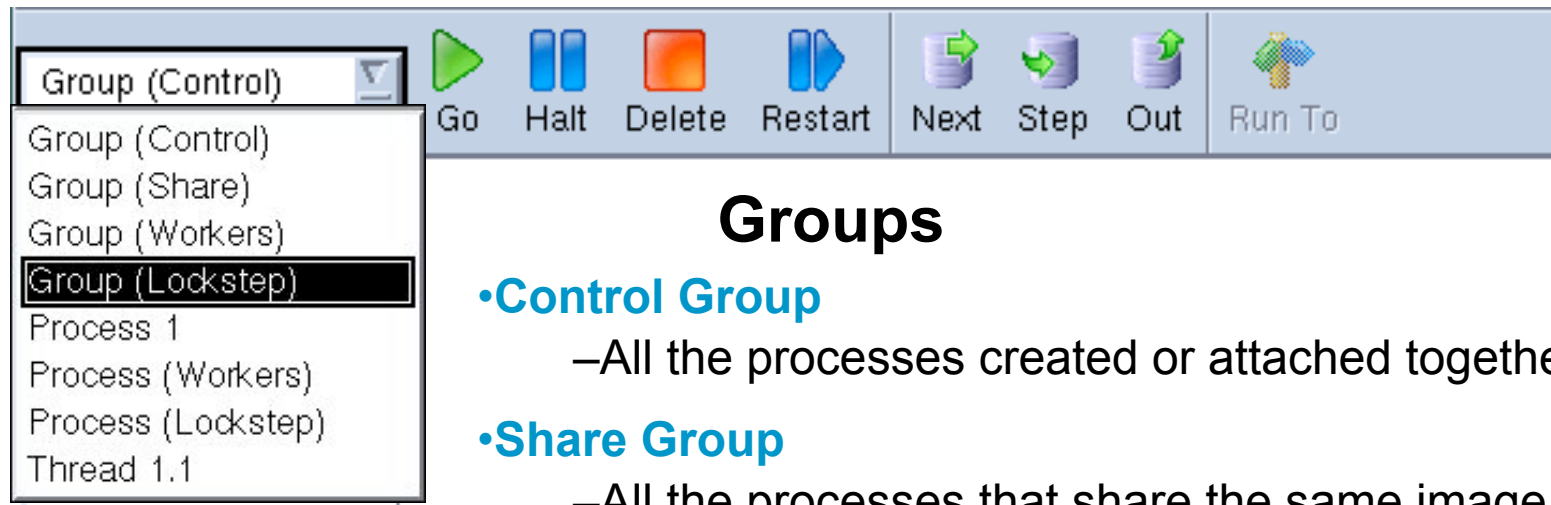
- There are more degrees of freedom for program control. TotalView gives you a full set of features to manage this complexity.
- It is important to understand how the different classes of commands and features work so as to avoid confusion.



Process Control Concepts

- Each process window is always focused on a specific process.
- Process focus can be easily switched
 - P+/P-, Dive in Root window and Process tab
- Processes can be 'held' - they will not run till unheld.
 - Process > Hold
- Breakpoints can be set to stop the process or the group
- Breakpoint and command scope can be simply controlled

Basic Process Control



Groups

- **Control Group**

- All the processes created or attached together

- **Share Group**

- All the processes that share the same image

- **Workers Group**

- All the processes or threads that are not recognized as manager or service processes or threads

- **Lockstep Group**

- All threads at the same PC

- **Call Graph Group**

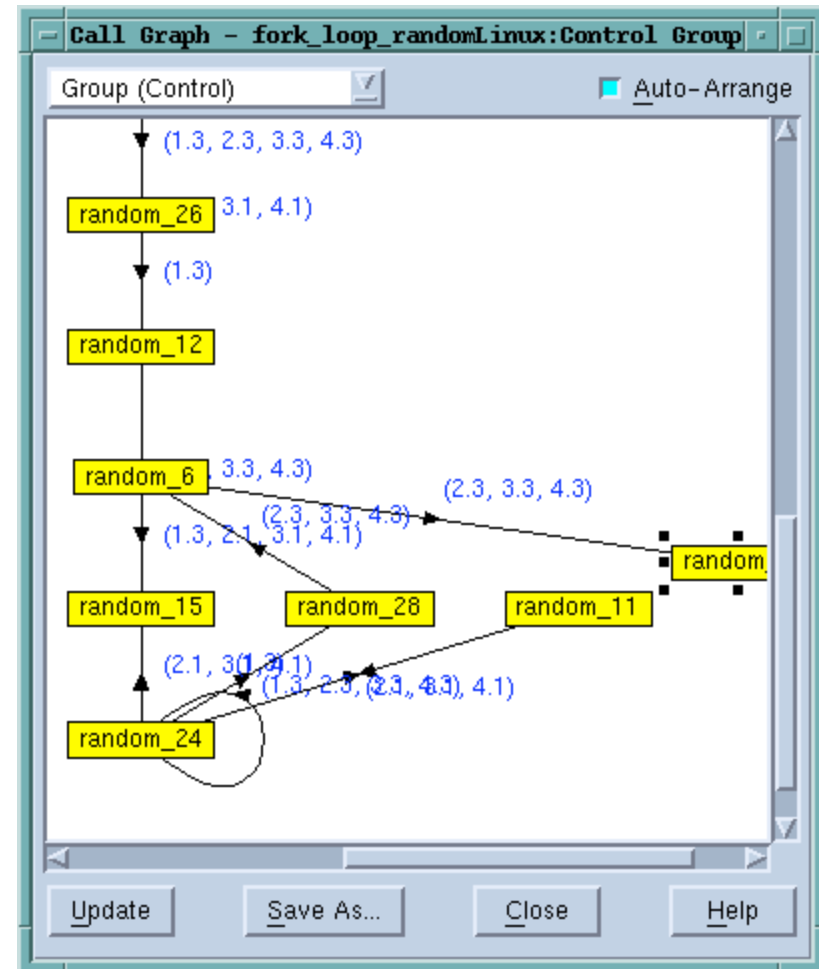
- All processes going through the same node in the call graph

- **User Defined Group**

- Process group defined in Custom Groups dialog

Call Graph

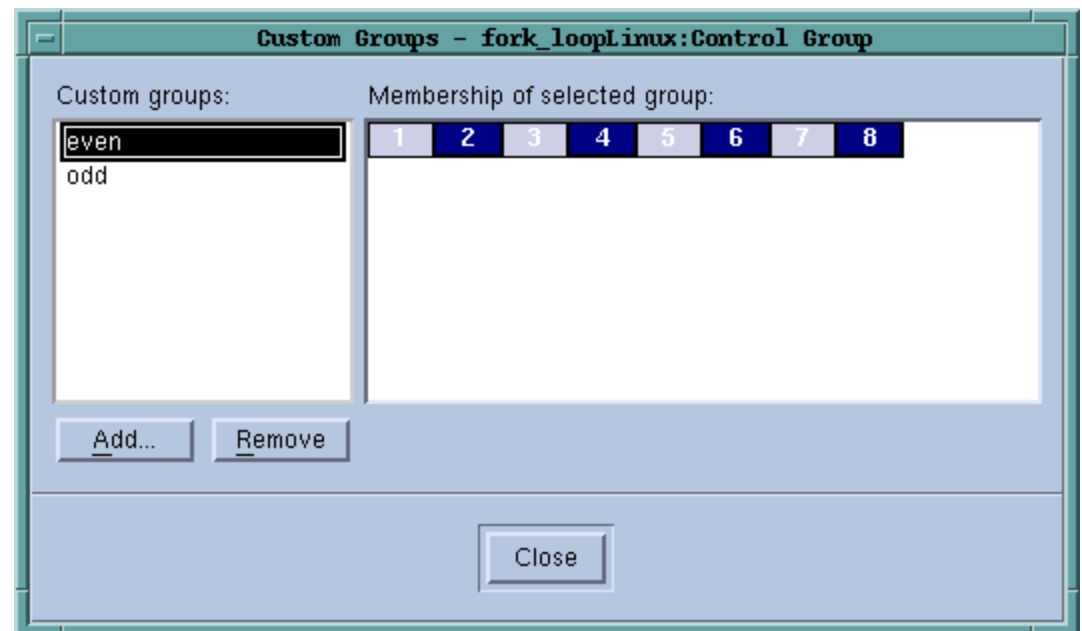
- **Quick view of program state**
 - Each call stack is a path
 - Functions are nodes
 - Calls are edges
 - Labeled with the MPI rank
 - Construct process groups
- **Look for outliers**



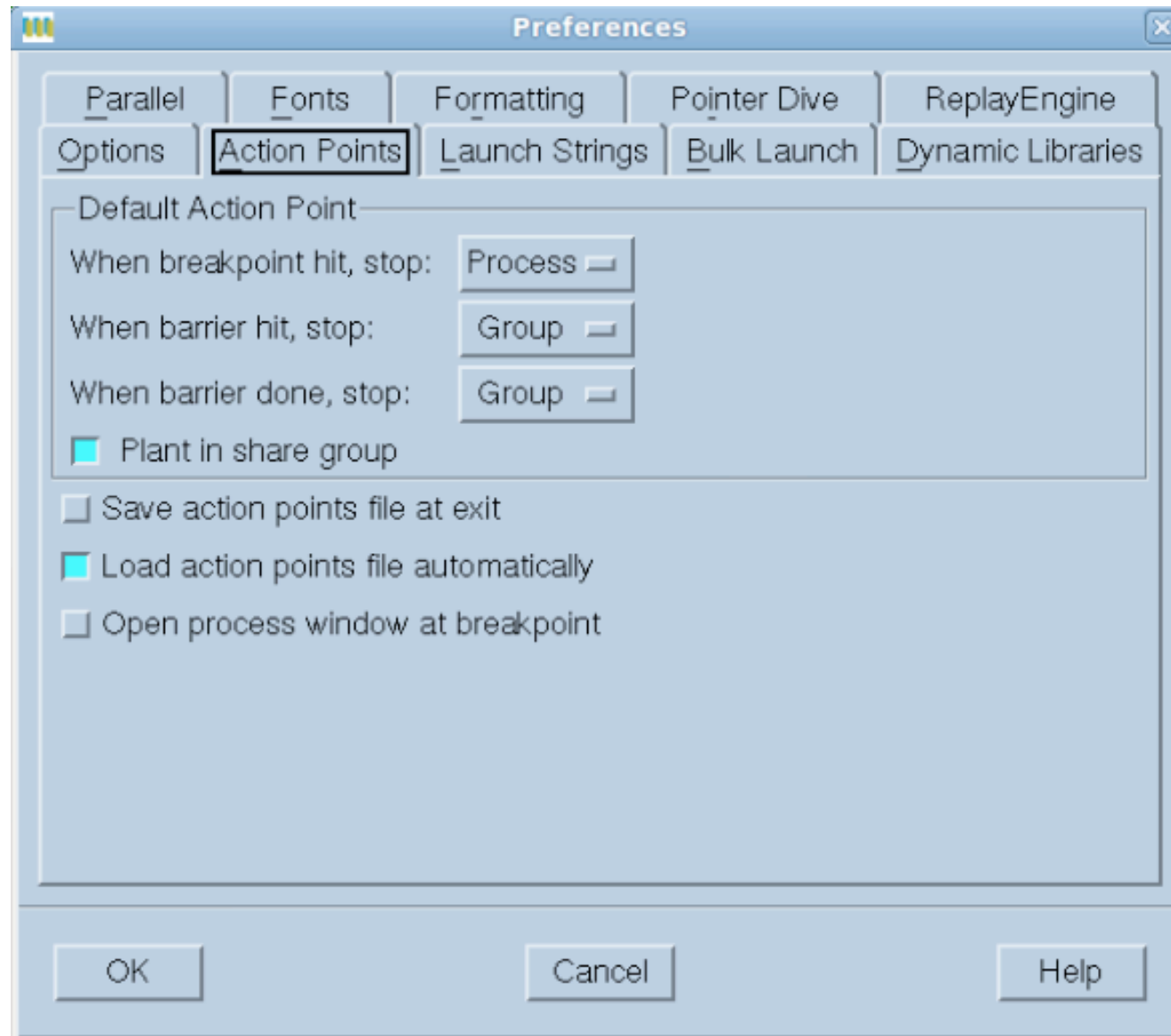
Dive on a node in the call graph to create a Call Graph group.

User Defined Groups

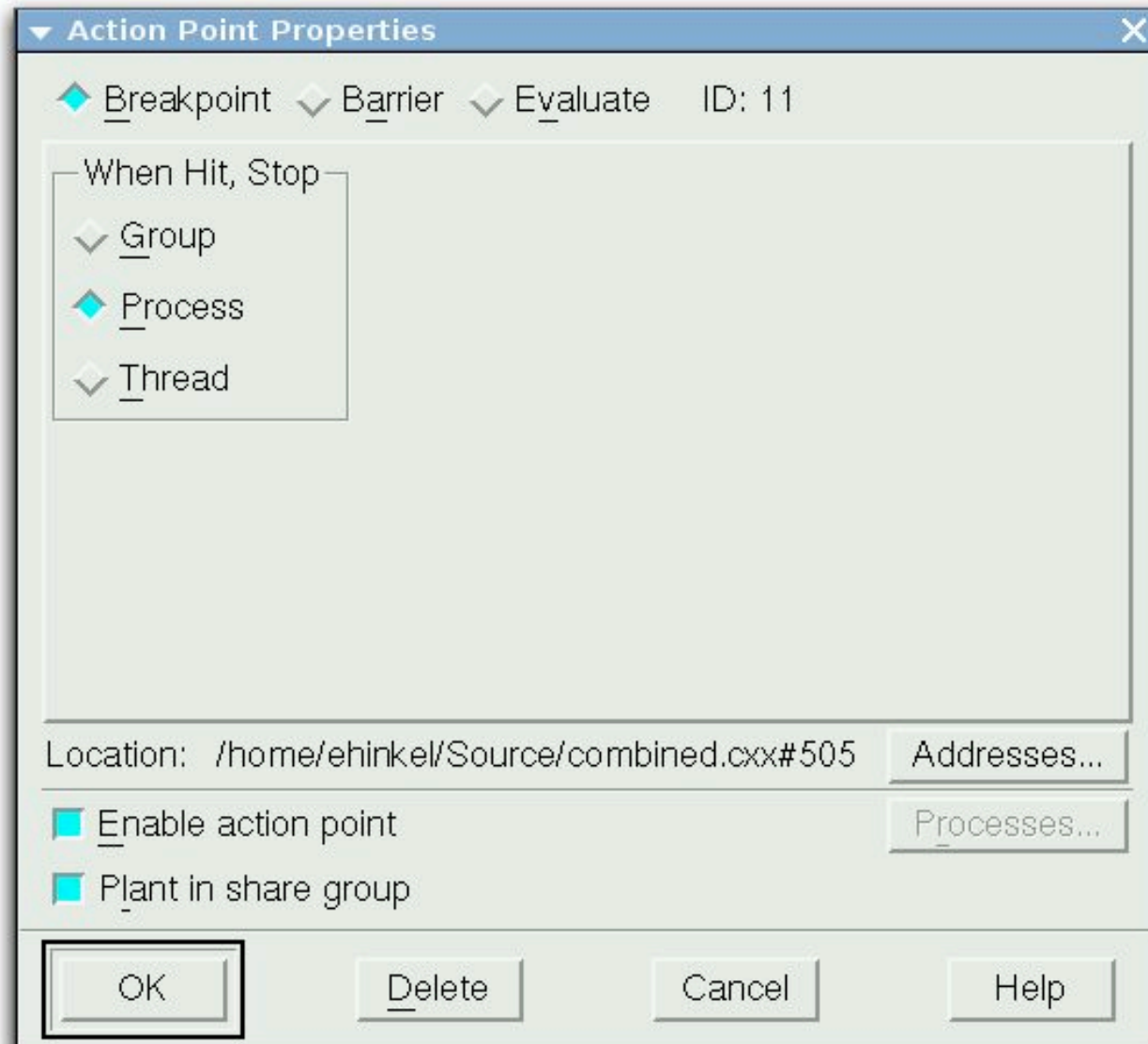
- **Group > Custom Groups**, to create a process group of some other specification
- **Group Membership** shown in **Processes Tab**
- **User defined groups** appear in the “Go” drop-down menu



Asynchronous Action Point Preferences



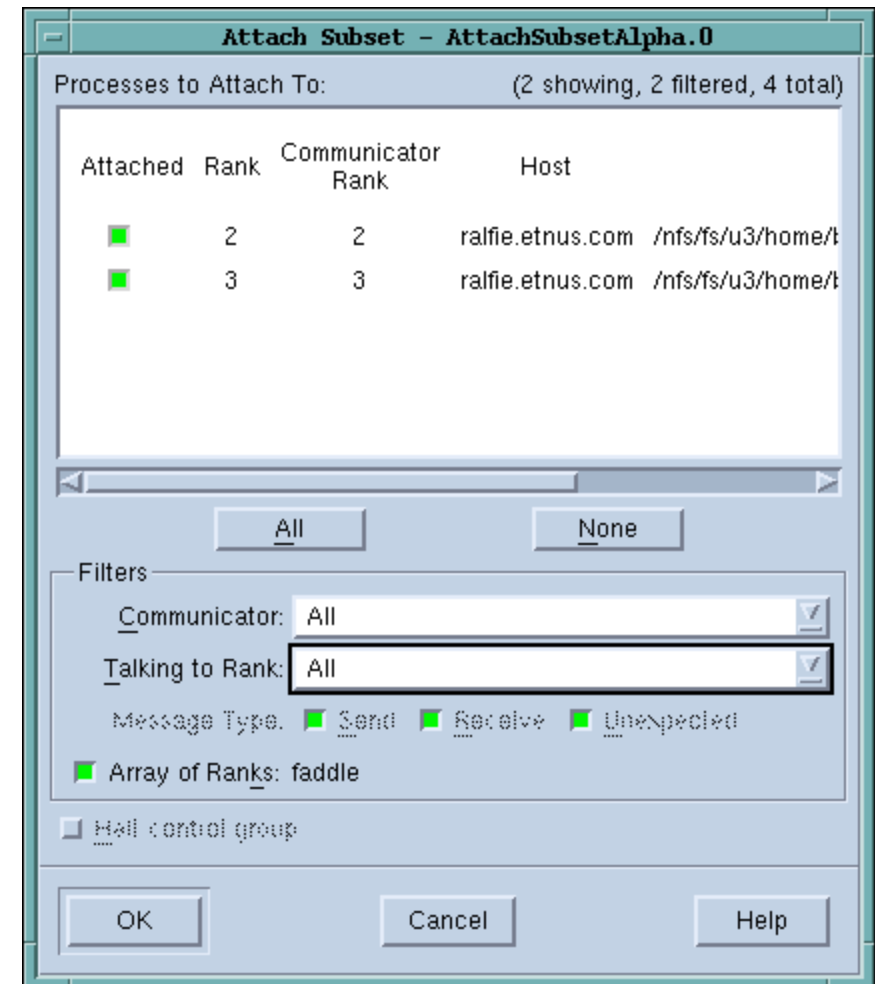
Asynchronous Action Point Control



Subset Attach

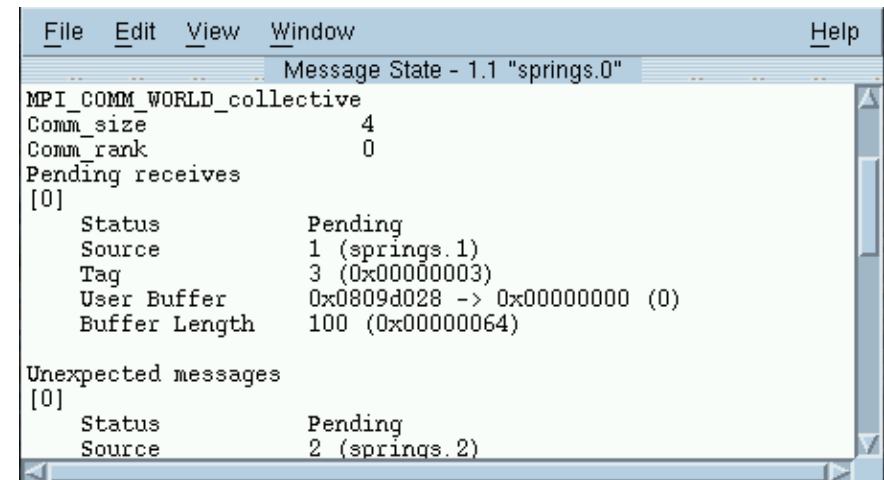
TotalView does not need to be attached to the entire job

- You can be attached to different subsets at different times through the run
- You can attach to a subset, run till you see trouble and then 'fan out' to look at more processes if necessary.
- This greatly reduces overhead
- It also requires a smaller license if you have a TotalView Team license.



View MPI Message Queues

- Information visible whenever MPI rank processes are halted
- Provides information from the MPI layer
 - Unexpected messages
 - Pending Sends
 - Pending Receives
- Use this info to debug
 - Deadlock situations
 - Load balancing
- May need to be enabled in the MPI library
 - --enable-debug



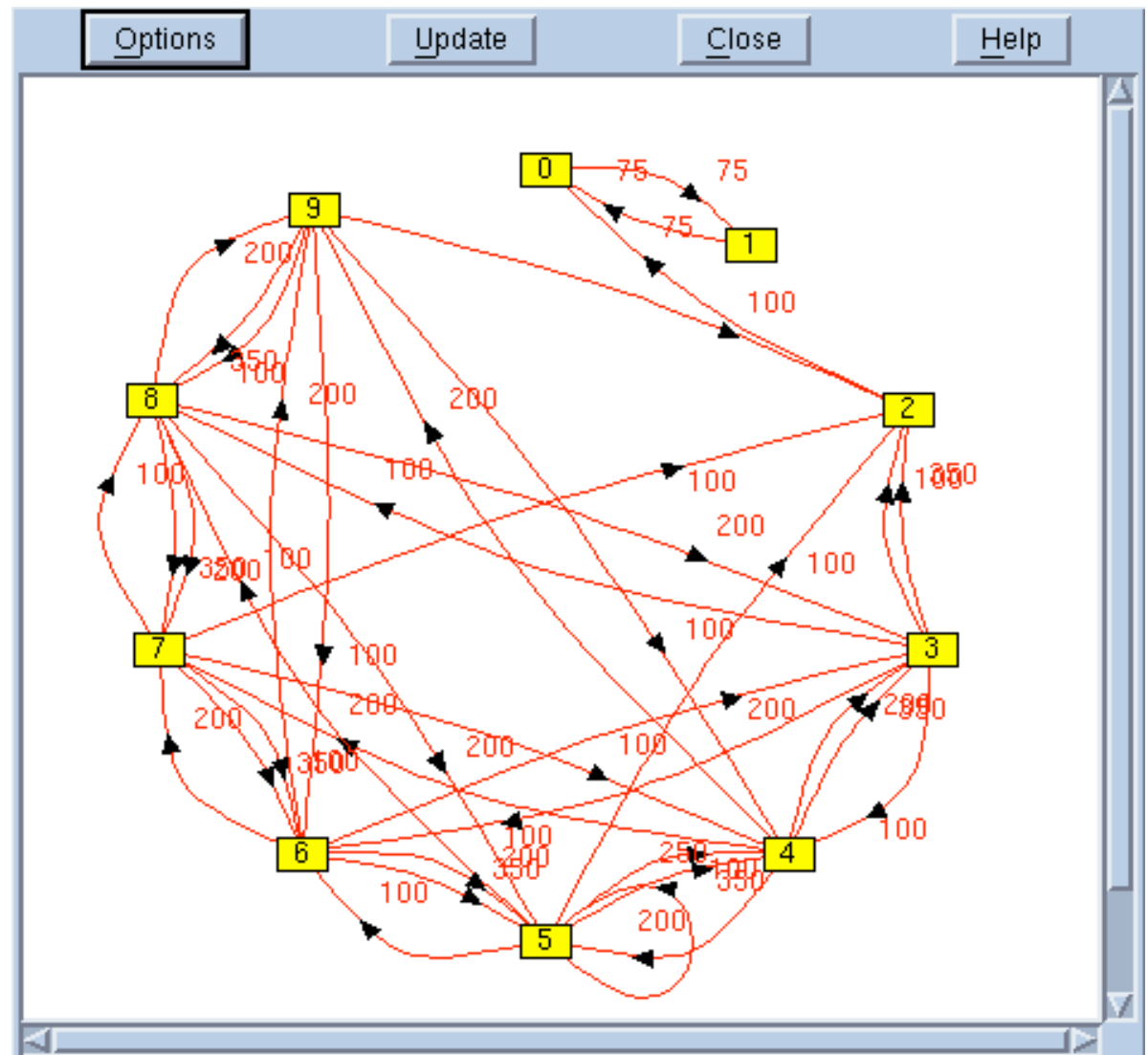
```

File Edit View Window Help
Message State - 1.1 "springs.0"
MPI_COMM_WORLD_collective
Comm_size      4
Comm_rank      0
Pending receives
[0]
  Status        Pending
  Source        1 (springs.1)
  Tag           3 (0x00000003)
  User Buffer    0x0809d028 -> 0x00000000 (0)
  Buffer Length  100 (0x00000064)

Unexpected messages
[0]
  Status        Pending
  Source        2 (springs.2)
  
```

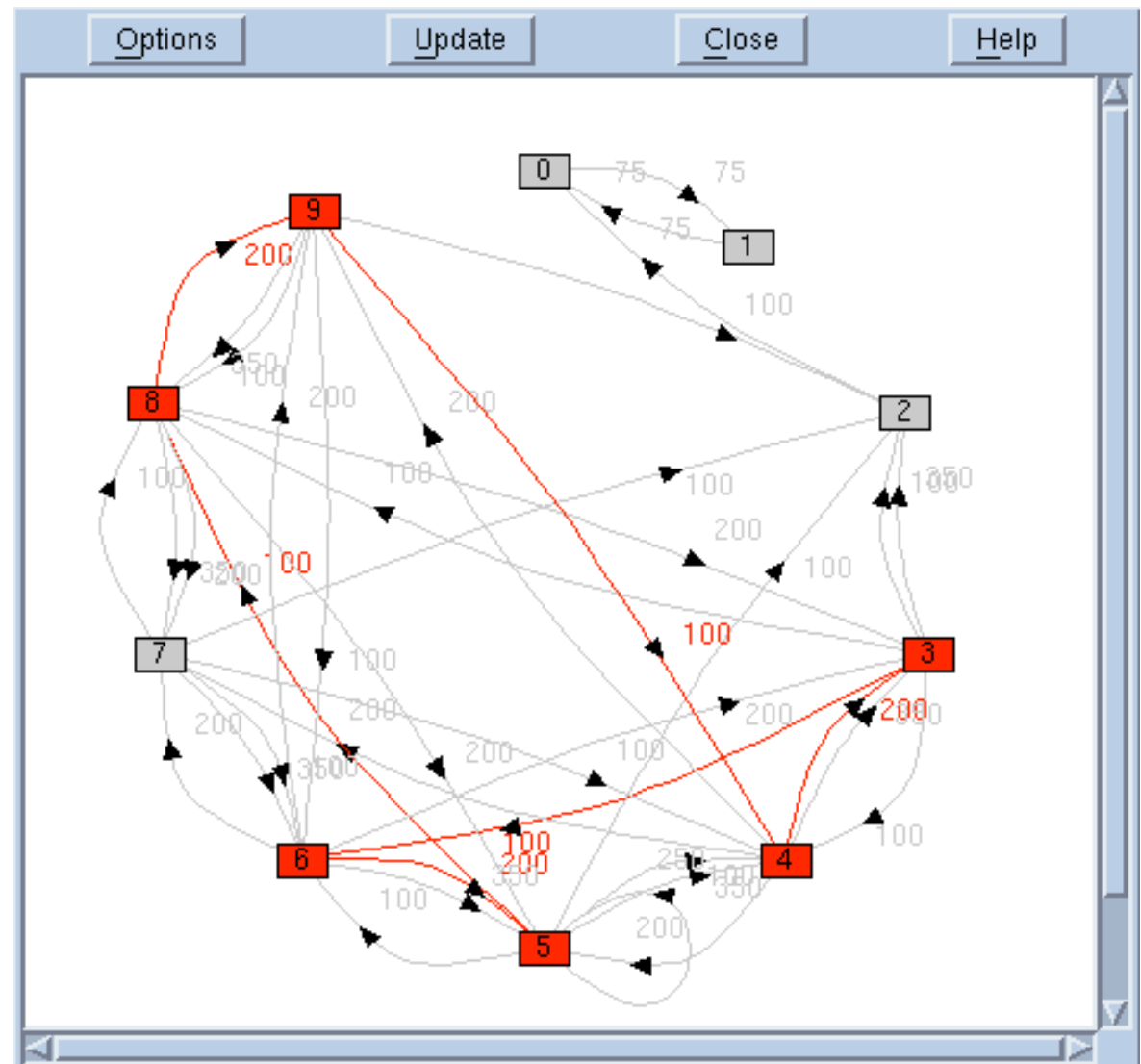
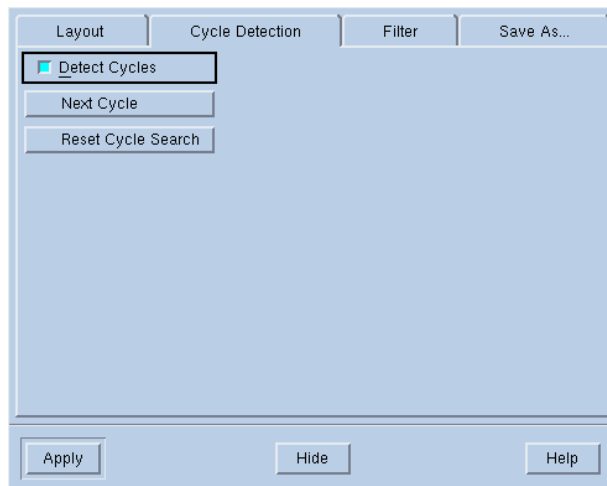
Message Queue Graph

- Hangs & Deadlocks
- Pending Messages
 - Receives
 - Sends
 - Unexpected
- Inspect
 - Individual entries
- Patterns



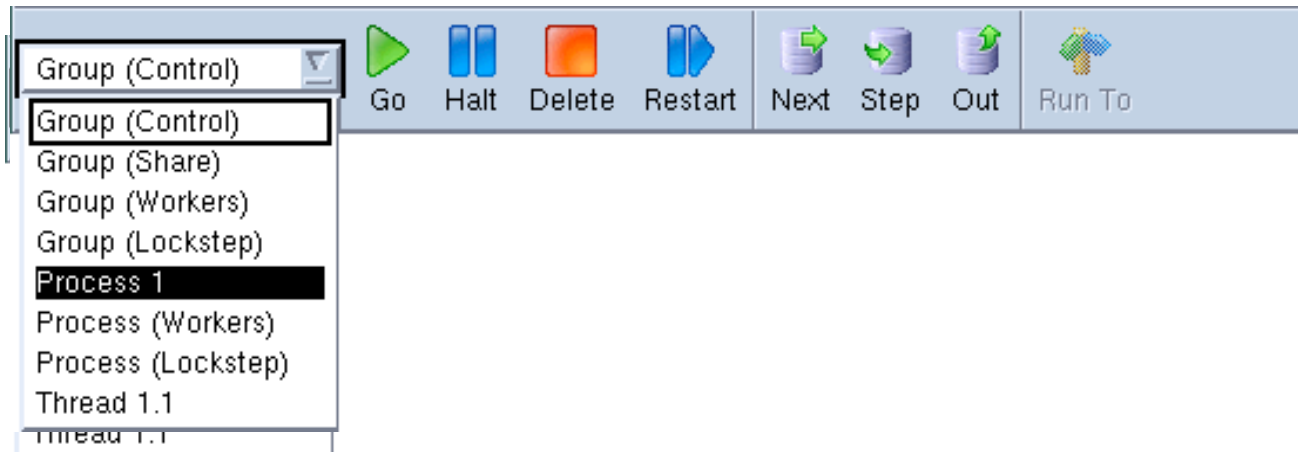
Message Queue Debugging

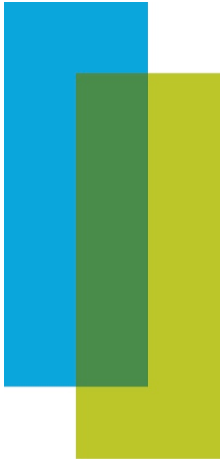
- **Filtering**
 - Tags
 - MPI Communicators
- **Cycle detection**
 - Find deadlocks



Strategies for Large Jobs: Focus Effort

- **Problem:** Some debugger operations are much more intensive than others, when multiplied by N this can add up.
 - **Strategy:** Reduce the interaction between the debugger and the processes
 - **Technique:** Use TotalView's process control features to
 - Single step one or a small set of processes rather than all of them
Use **Group > Custom Group** to create named groups
 - Use RunTo or Breakpoints to control large groups of processes
 - Use Watchpoints to watch for variable changes





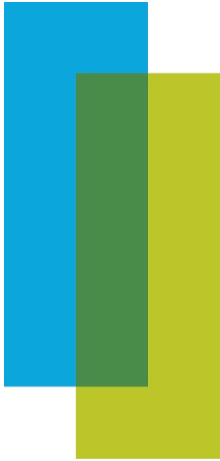
Strategies for Large Jobs

- **Reduce N**

- **Problem:** Each process added requires overhead
- **Strategy:** Reduce the number of processes TotalView is attached to
 - Simply reducing N is best, however data or algorithm may require large N
- **Technique:** subset attach mechanism

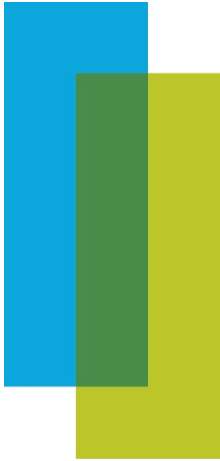
- **Focus Effort**

- **Problem:** Some debugger operations are much more intensive than others, and when multiplied by N this could be significant
- **Strategy:** Reduce the interaction between the debugger and the processes
- **Technique:** Use TotalView's process control features to
 - Avoid single stepping
 - Focus on one or a small set of processes



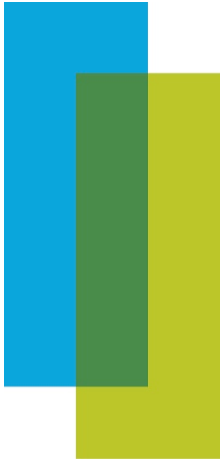
TotalView

Memory Debugging



What is a Memory Bug?

- **A Memory Bug is a mistake in the management of heap memory**
 - Failure to check for error conditions
 - Leaking: Failure to free memory
 - Dangling references: Failure to clear pointers
 - Memory Corruption
 - Writing to memory not allocated
 - Over running array bounds

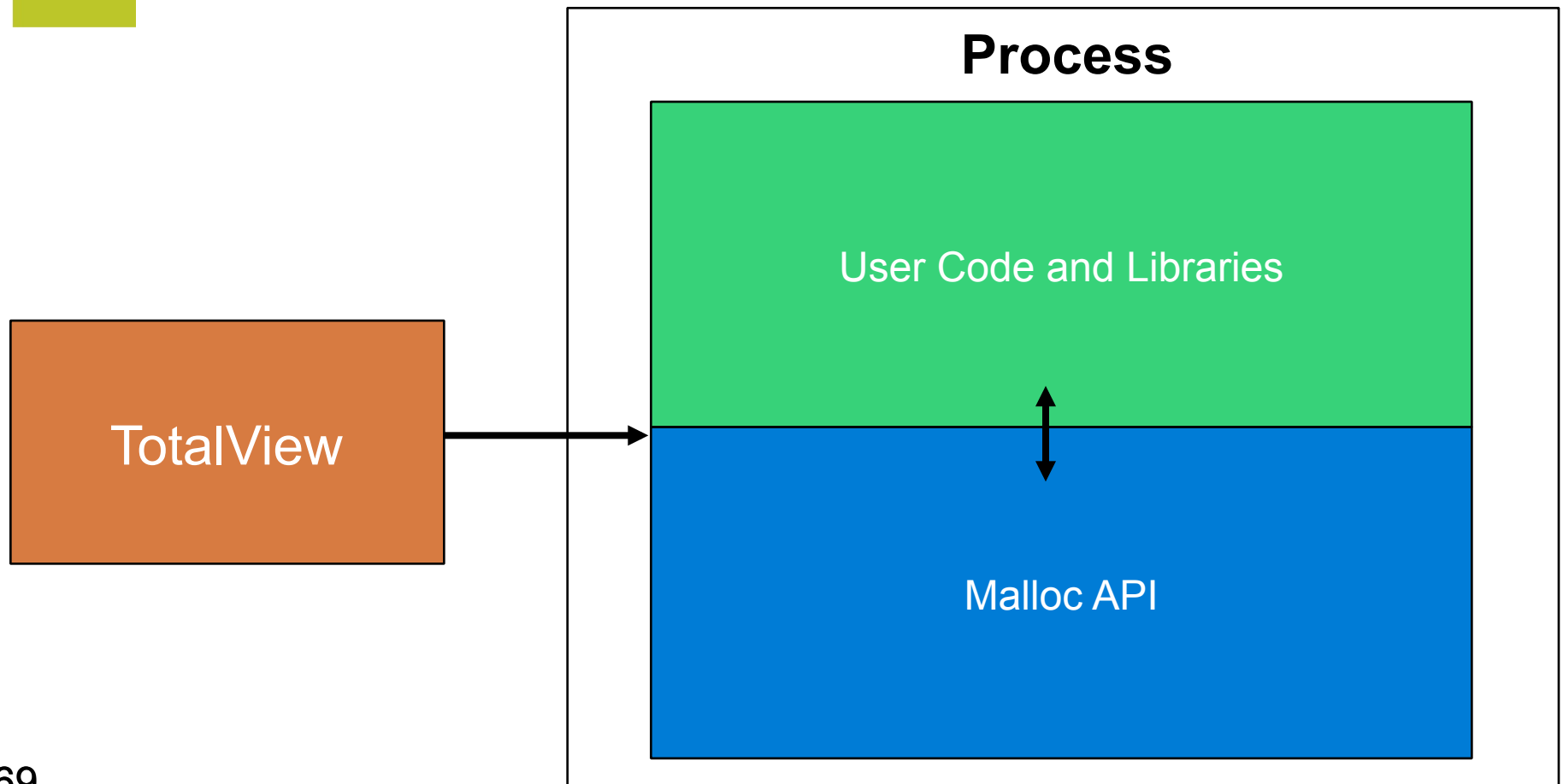


What is a Memory Bug?

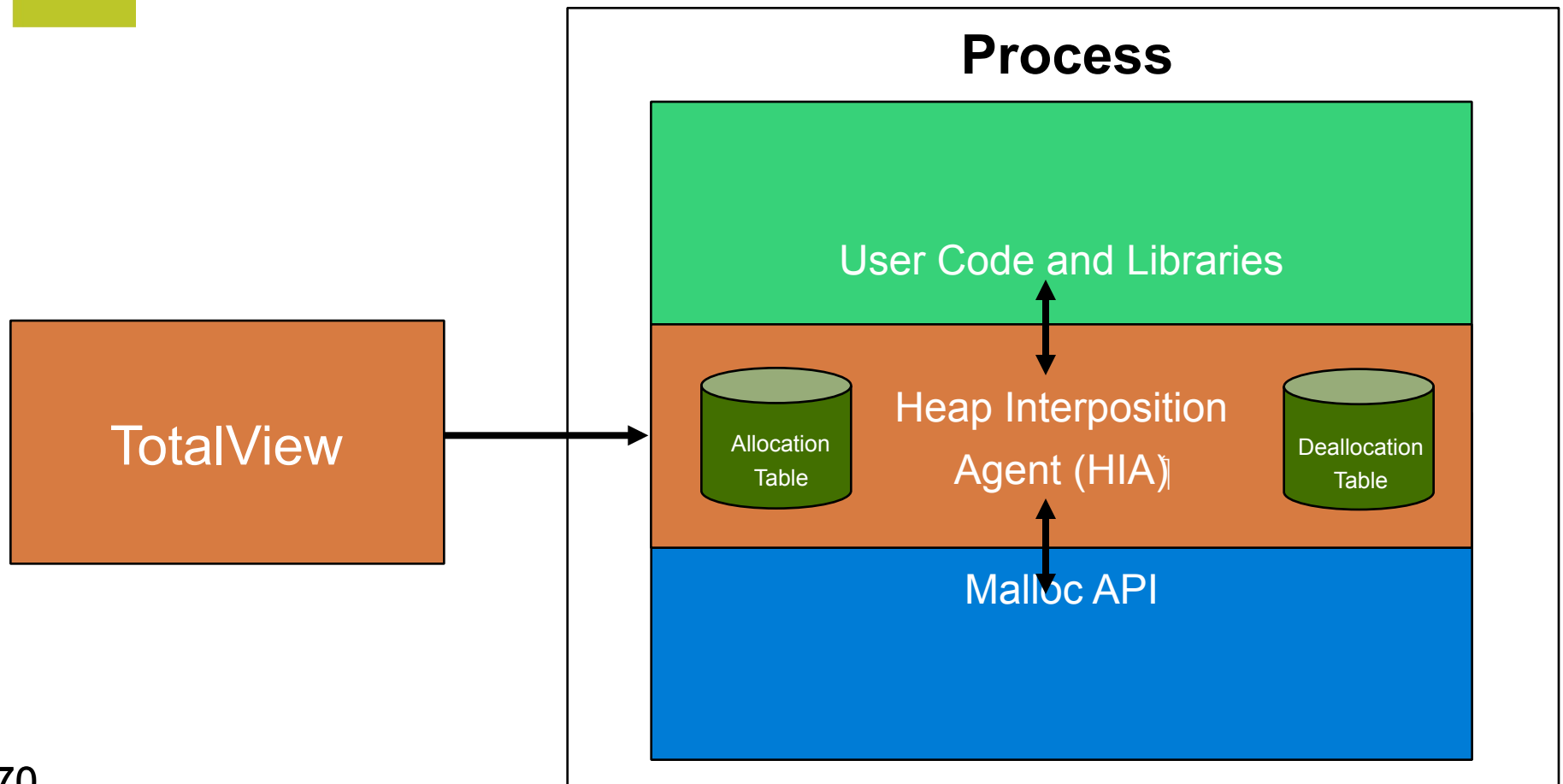
Why Are Memory Bugs Hard to Find?

- Memory problems can lurk
 - For a given scale or platform or problem, they may not be fatal
 - Libraries could be source of problem
 - The fallout can occur at any subsequent memory access through a pointer
 - The mistake is rarely fatal in and of itself
 - The mistake and fallout can be widely separated
- Potentially 'racy'
 - Memory allocation pattern non-local
 - Even the fallout is not always fatal. It can result in data corruption which may or may not result in a subsequent crash
- May be caused by or cause of a 'classic' bug

The Agent and Interposition

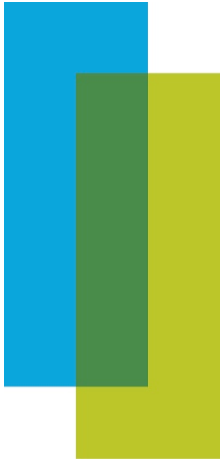


The Agent and Interposition



TotalView HIA Technology

- **Advantages of TotalView HIA Technology**
 - Use it with your existing builds
 - No Source Code or Binary Instrumentation
 - Programs run nearly full speed
 - Low performance overhead
 - Low memory overhead
 - Efficient memory usage
 - Support wide range of platforms and compilers

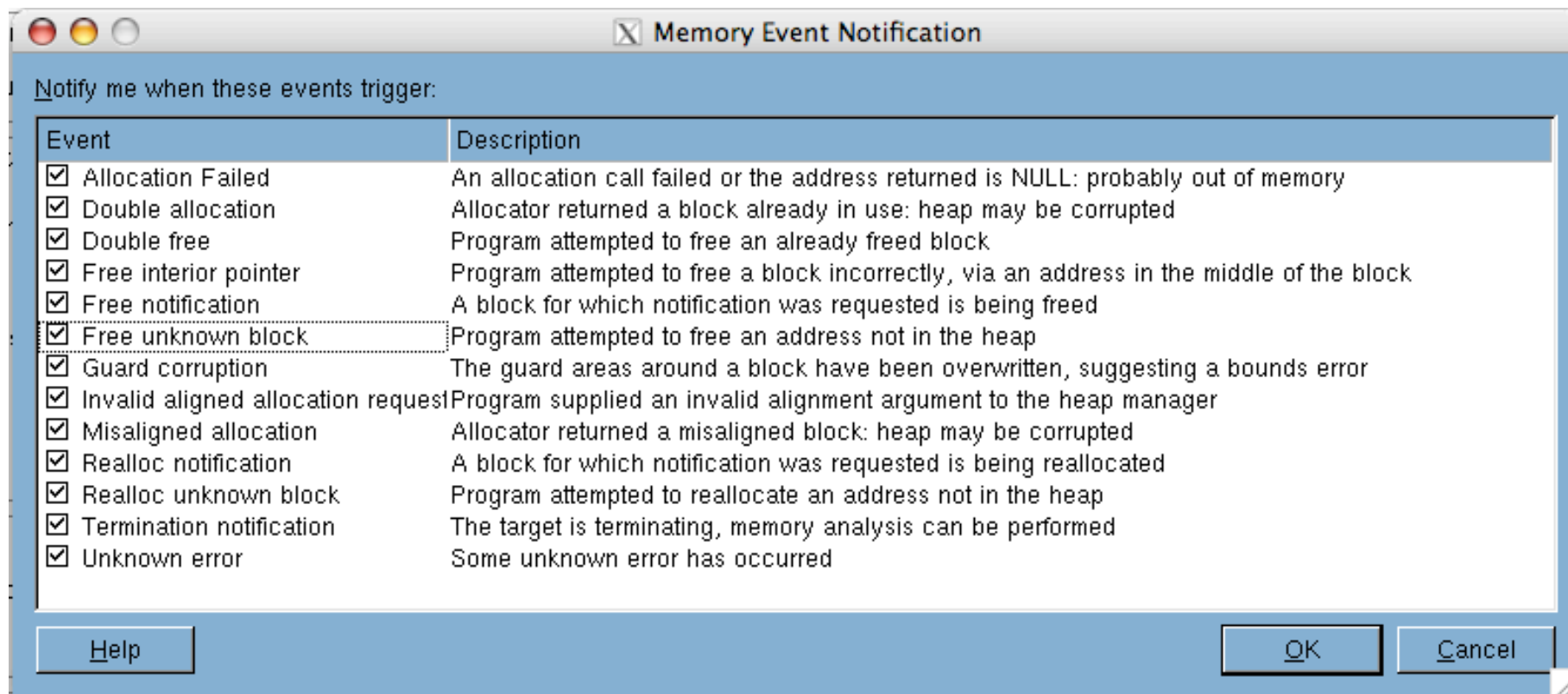


Memory Debugger Features

- Automatically detect allocation problems
- View the heap
- Leak detection
- Block painting
- Memory Hoarding
- Dangling pointer detection
- Deallocation/reallocation notification
- Memory Corruption Detection - Guard Blocks
- Memory Comparisons between processes
- Collaboration features

Enabling Memory Debugging

Memory Event Notification




Notify me when these events trigger:

| Event | Description |
|--|--|
| <input checked="" type="checkbox"/> Allocation Failed | An allocation call failed or the address returned is NULL: probably out of memory |
| <input checked="" type="checkbox"/> Double allocation | Allocator returned a block already in use: heap may be corrupted |
| <input checked="" type="checkbox"/> Double free | Program attempted to free an already freed block |
| <input checked="" type="checkbox"/> Free interior pointer | Program attempted to free a block incorrectly, via an address in the middle of the block |
| <input checked="" type="checkbox"/> Free notification | A block for which notification was requested is being freed |
| <input checked="" type="checkbox"/> Free unknown block | Program attempted to free an address not in the heap |
| <input checked="" type="checkbox"/> Guard corruption | The guard areas around a block have been overwritten, suggesting a bounds error |
| <input checked="" type="checkbox"/> Invalid aligned allocation request | Program supplied an invalid alignment argument to the heap manager |
| <input checked="" type="checkbox"/> Misaligned allocation | Allocator returned a misaligned block: heap may be corrupted |
| <input checked="" type="checkbox"/> Realloc notification | A block for which notification was requested is being reallocated |
| <input checked="" type="checkbox"/> Realloc unknown block | Program attempted to reallocate an address not in the heap |
| <input checked="" type="checkbox"/> Termination notification | The target is terminating, memory analysis can be performed |
| <input checked="" type="checkbox"/> Unknown error | Some unknown error has occurred |

Help OK Cancel

Memory Event Details Window

Memory Event Details - free_doubleLinux - 2.1

 Program attempted to deallocate an already deallocated block

Block Information

0x08049858 171 bytes 0x08049903

Status: Deallocated
Flags: Hoarded

Block Backtrace Information

Select the desired tab below to see the block allocation or deallocation backtrace. Backtrace information may not always be available. Examine the Process Window to see the point at which the application stopped due to the event.

Backtrace

| ID | Function | Line # | Source Information |
|----|-------------------------|--------|---|
| | TV_HEAP_free_interposer | 3010 | /home/barryk/bld/linux-x86/interpos... |
| | free | 167 | /home/barryk/bld/linux-x86/interpos... |
| | main | 35 | /home/barryk/tests/free_double_free... |
| | __libc_start_main | | /lib/1686/libc.so.6 |
| | _start | | /nfs/netapp/u2/home/barryk/tests/fre... |

Source /home/barryk/tests/free_double_free.c

```
30 printf ("malloced %4d (%#6x) bytes at %p\n", region_size, region_size, s );
31
32 /* now release the memory */
33
34 printf ("free ( %p )      [correct usage]\n", s );
35 free ( s );
```

Point of Allocation Point of Deallocation

Close View in Block Properties window Help


Memory Block Properties


Memory Blocks

0x08049858 - 0x08049903

0x08049858 171 bytes

Block Information

Status:  The memory block has an *allocated* status.

Point of Allocation:  Function: TV_STACK_get_backtrace (PC=0x40027e68)
File: /home/barryk/bld/linux-x86/interposition/src/backtrace/backtr...
Line: 24

Block Flags

☐ Notify when deallocated
☐ Notify when reallocated

Block Backtrace Information

Backtrace

| ID | Function | Line # | Source Information |
|----|---------------------------|--------|--|
| -1 | TV_STACK_get_backtrace | 24 | /home/barryk/bld/linux-x86/interpositio... |
| | insert_backtrace | 594 | /home/barryk/bld/linux-x86/interposition/sr... |
| | check_allocation | 1117 | /home/barryk/bld/linux-x86/interposition/sr... |
| | malloc_body | 2122 | /home/barryk/bld/linux-x86/interposition/sr... |
| | TV_HEAP_malloc_interposer | 2152 | /home/barryk/bld/linux-x86/interposition/sr... |
| | malloc | 149 | /home/barryk/bld/linux-x86/interposition/sr... |
| | main | 27 | free_double_free.c |

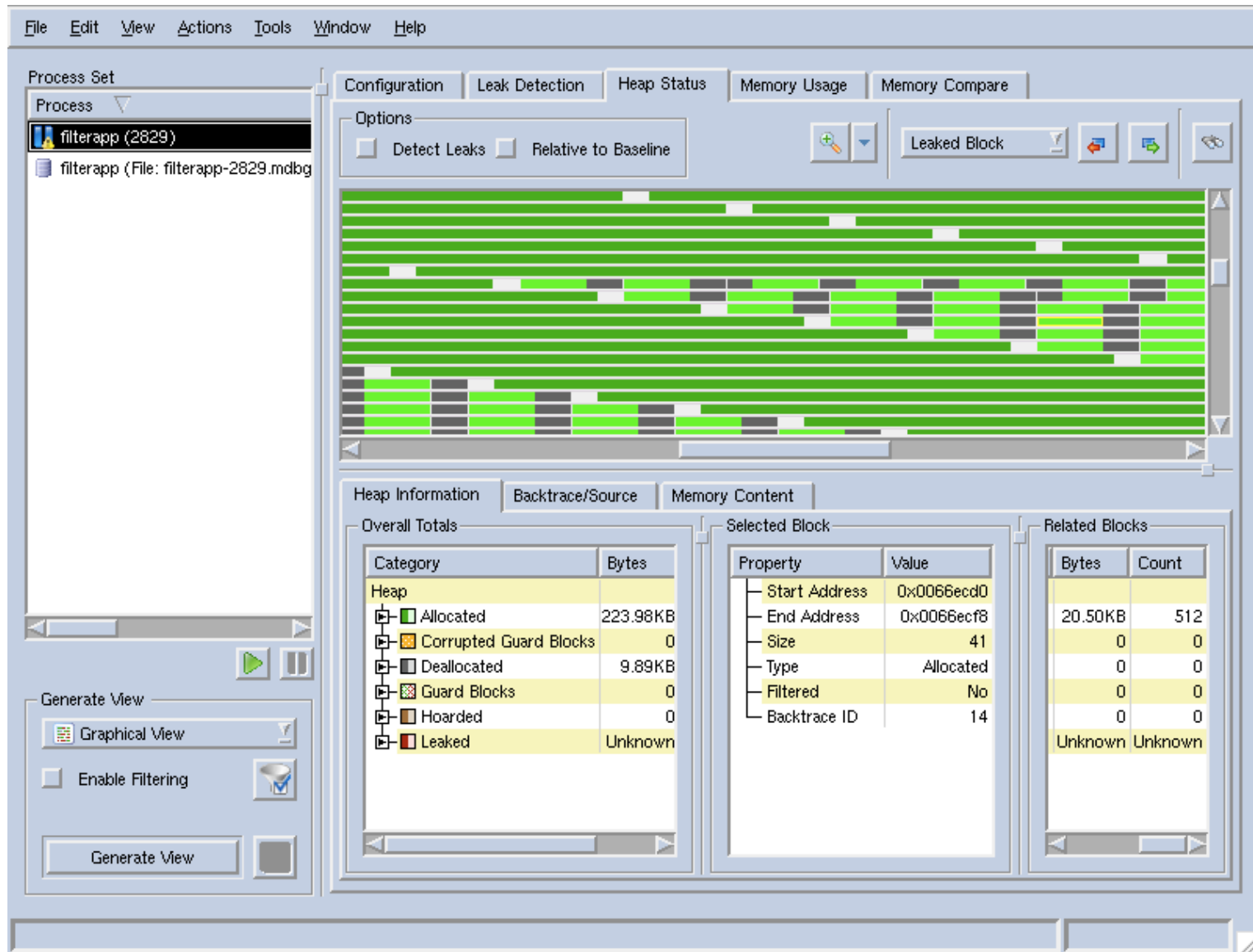
Source /home/barryk/bld/linux-x86/interposition/src/backtrace/backtrace_glibc.c

```
19 int
20 TV_STACK_get_backtrace ( void **pcs, int vector_length )
21 {
22     size_t ret_val;
```

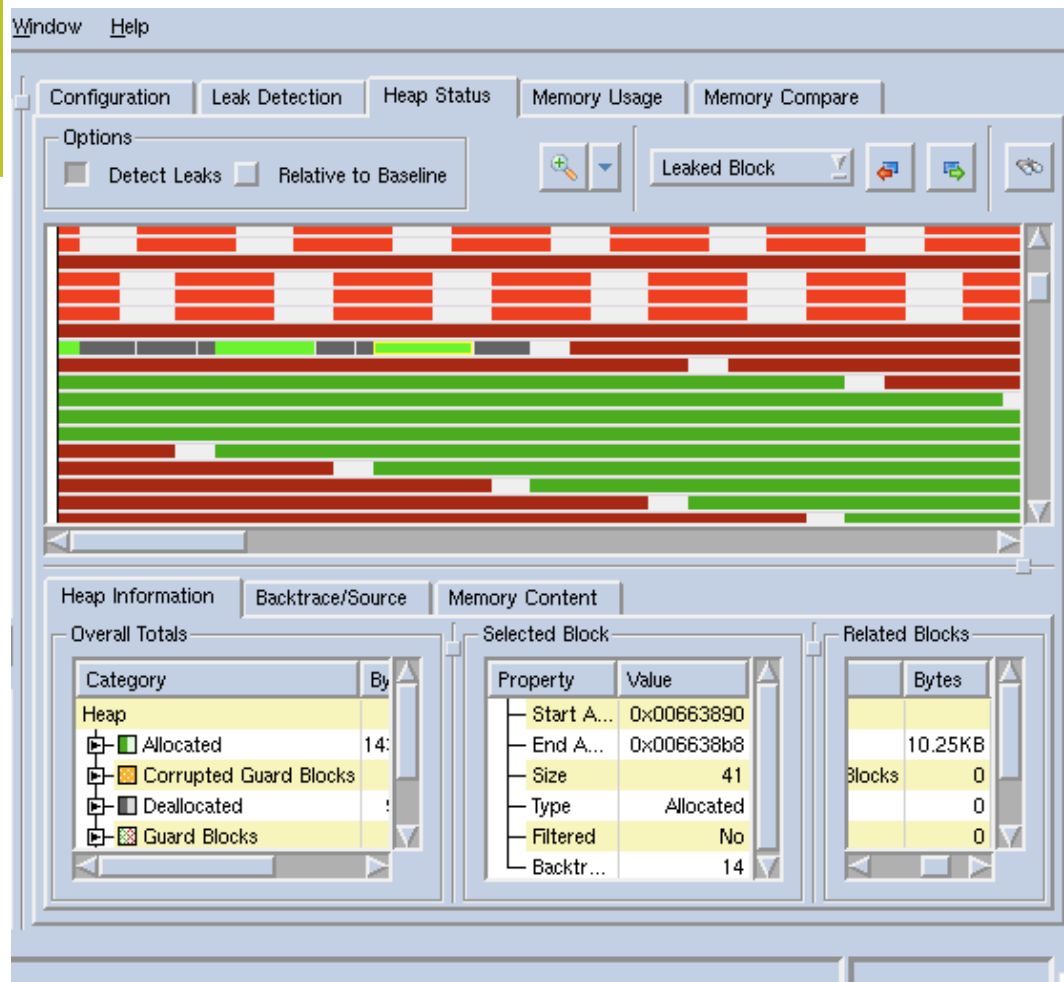
Point of Allocation Point of Deallocation

Close Hide Backtrace Information Help

Heap Graphical View

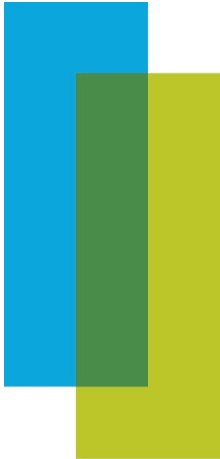


Leak Detection

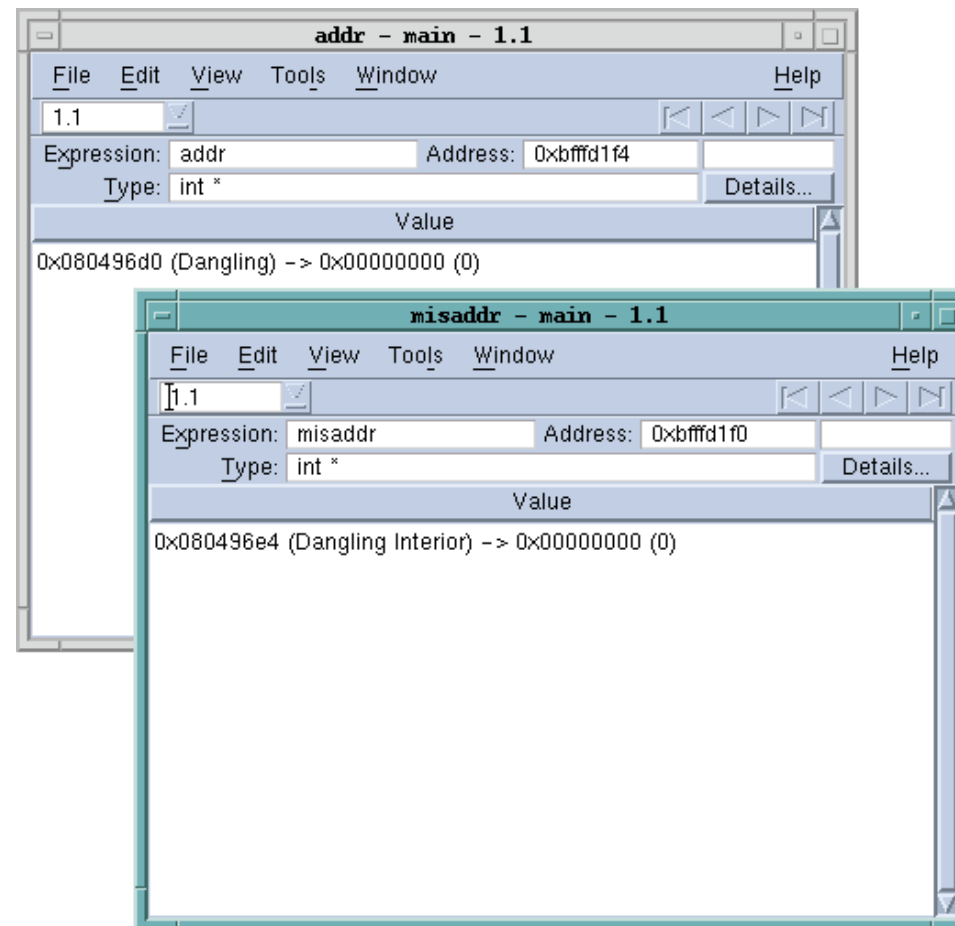


Leak Detection

- **Based on Conservative Garbage Collection**
- **Can be performed at any point in runtime**
 - **Helps localize leaks in time**
- **Multiple Reports**
 - **Backtrace Report**
 - **Source Code Structure**
 - **Graphically Memory Location**



Dangling Pointer Detection



Memory Corruption Detection (Guard Blocks)

Guard Blocks

On ☒ Guard Blocks

Pre-Guard Size: 8 bytes

Pattern: 0x77777777

Post-Guard Size: 8 bytes

Pattern: 0x99999999

Maximum Guard Size: 0 bytes

Configuration Leak Detection Heap Status Memory Usage Memory Compare

| | Preceding Block | Corrupted Block | Following Block |
|---|-------------------------------------|------------------------------------|-----------------|
| 1 | 0x0804a028 - 171 bytes - 0x0804a0d2 | 0x0804a0e8 - 32 bytes - 0x0804a107 | |

Backtrace

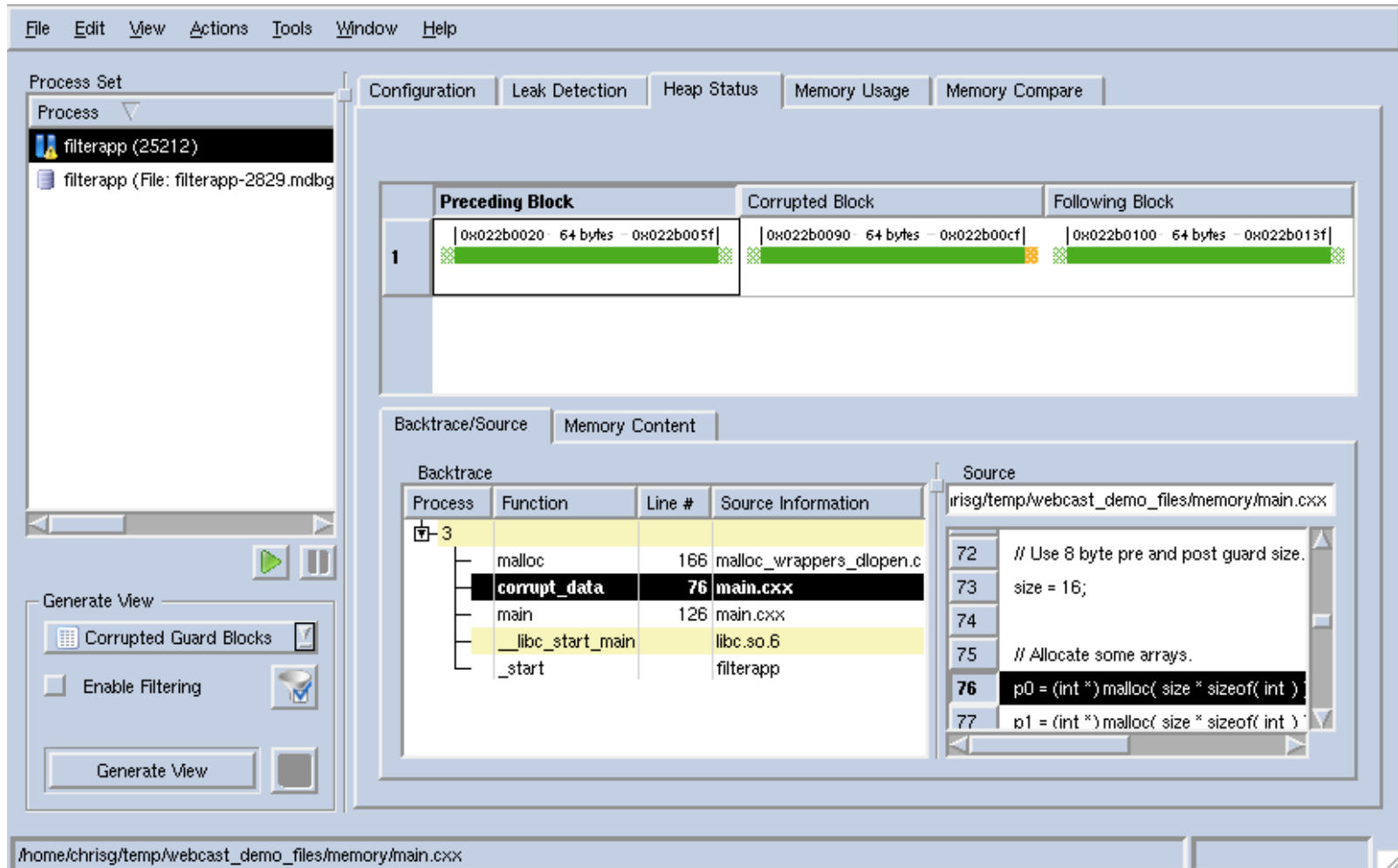
| Process | Function | Line # | Source Information |
|---------|-------------------|------------|---------------------|
| 1 | malloc | 149 | malloc_wrapped |
| | main | 106 | hla_events.c |
| | __libc_start_main | | libc.so.6 |
| | _start | | hla_events |

Source

```

/home/barryk/fred/hla_events.c
101
102     break;
103
104     case notify_guard_corrupti
105
106     s3 = (char *)malloc ( 0xab
107
  
```

Memory Corruption Report



The screenshot displays the TotalView Technologies Memory Corruption Report interface. The main window is titled "Process Set" and shows a list of processes, including "filterapp (25212)". The "Configuration" tab is selected, showing details for a corrupted memory block. The "Corrupted Block" is located at address 0x022b0090, 64 bytes in size, and is marked as corrupted. The "Preceding Block" is at 0x022b0020, 64 bytes, and the "Following Block" is at 0x022b0100, 64 bytes. The "Backtrace/Source" tab is also visible, showing a stack trace of the program's execution. The stack trace includes the following entries:

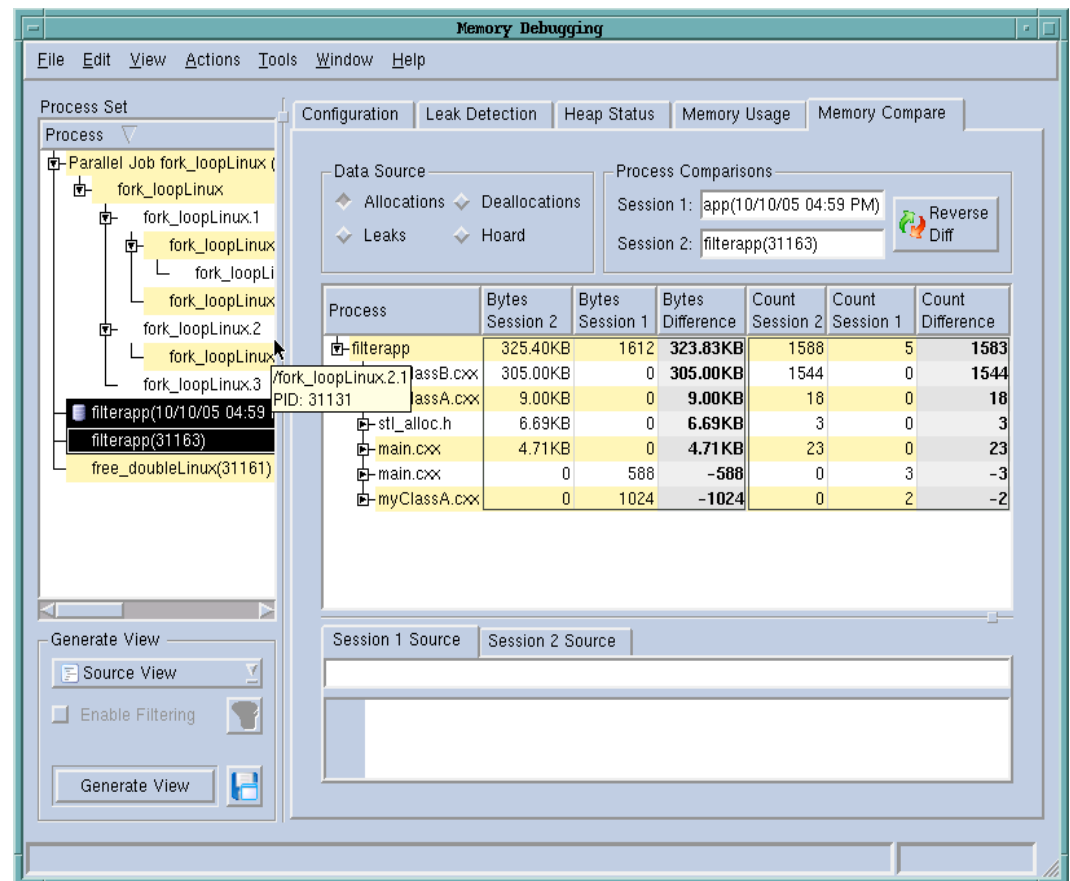
| Process | Function | Line # | Source Information |
|---------|---------------------|-----------|--------------------------|
| 3 | malloc | 166 | malloc_wrappers_dlopen.c |
| | corrupt_data | 76 | main.cxx |
| | main | 126 | main.cxx |
| | _libc_start_main | | libc.so.6 |
| | _start | | filterapp |

The "Source" tab shows the source code for the corrupted block, located at `/home/chrisg/temp/webcast_demo_files/memory/main.cxx`. The code snippet includes the following lines:

```
72 // Use 8 byte pre and post guard size.  
73 size = 16;  
74  
75 // Allocate some arrays.  
76 p0 = (int *) malloc( size * sizeof( int ) );  
77 p1 = (int *) malloc( size * sizeof( int ) );
```

Memory Comparisons

- **“Diff” live processes**
 - Compare processes across cluster
- **Compare with baseline**
 - See changes between point A and point B
- **Compare with saved session**
 - Provides memory usage change from last run



Memory Debugging

File Edit View Actions Tools Window Help

Process Set

- Parallel Job fork_loopLinux (
 - fork_loopLinux
 - fork_loopLinux.1
 - fork_loopLinux
 - fork_loopLinux
 - fork_loopLinux.2
 - fork_loopLinux.3
 - filterapp(10/10/05 04:59)
 - filterapp(31163)
 - free_doubleLinux(31161)

Configuration Leak Detection Heap Status Memory Usage **Memory Compare**

Data Source

- Allocations Deallocations
- Leaks Hoard

Process Comparisons

- Session 1: app(10/10/05 04:59 PM)
- Session 2: filterapp(31163)
- Reverse Diff

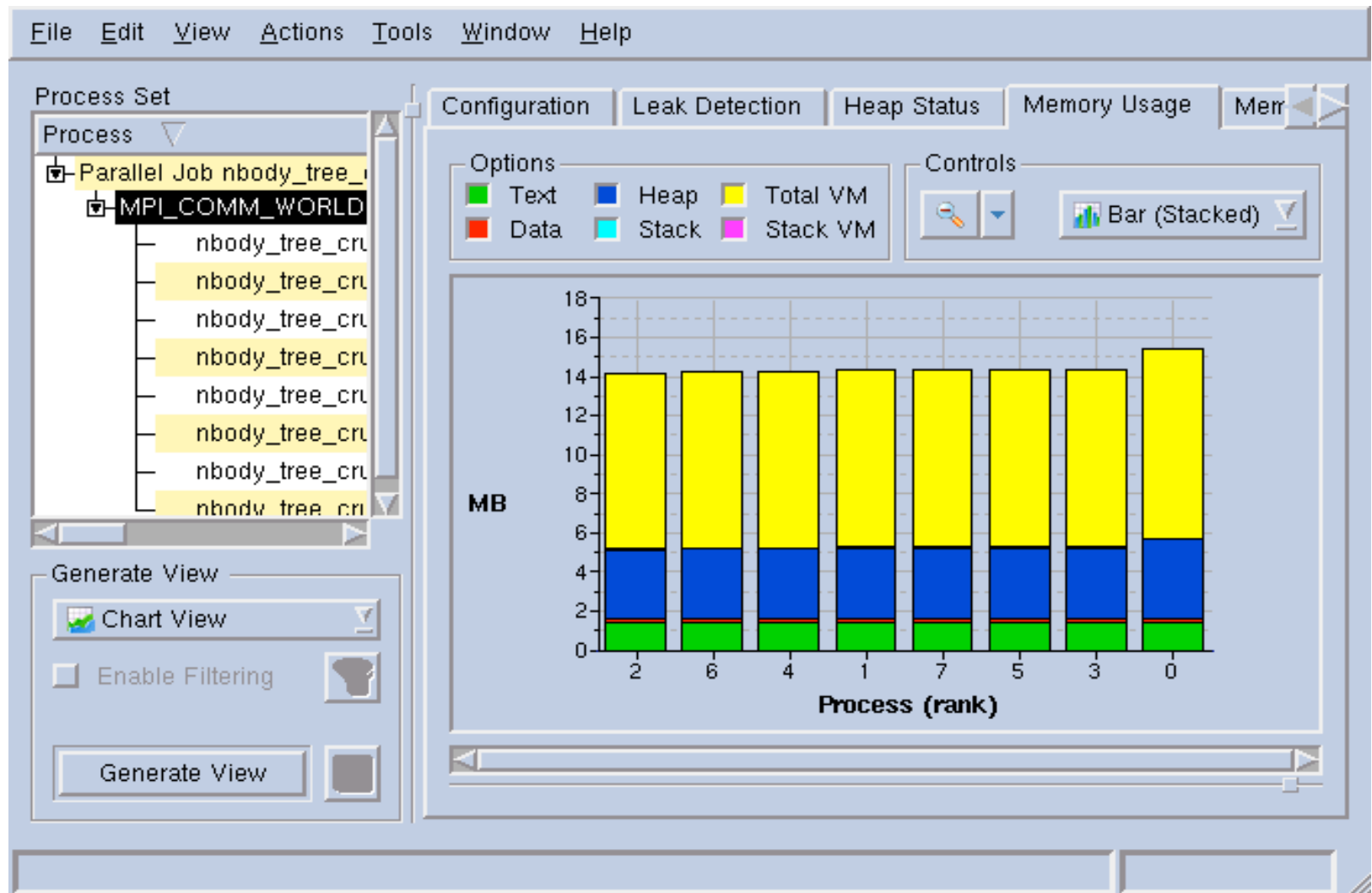
| Process | Bytes Session 2 | Bytes Session 1 | Bytes Difference | Count Session 2 | Count Session 1 | Count Difference |
|--------------------|-----------------|-----------------|------------------|-----------------|-----------------|------------------|
| filterapp | 325.40KB | 1612 | 323.83KB | 1588 | 5 | 1583 |
| fork_loopLinux.2.1 | 305.00KB | 0 | 305.00KB | 1544 | 0 | 1544 |
| main.cxx | 9.00KB | 0 | 9.00KB | 18 | 0 | 18 |
| stl_alloc.h | 6.69KB | 0 | 6.69KB | 3 | 0 | 3 |
| main.cxx | 4.71KB | 0 | 4.71KB | 23 | 0 | 23 |
| main.cxx | 0 | 588 | -588 | 0 | 3 | -3 |
| myClassA.cxx | 0 | 1024 | -1024 | 0 | 2 | -2 |

Generate View

- Source View
- Enable Filtering
- Generate View

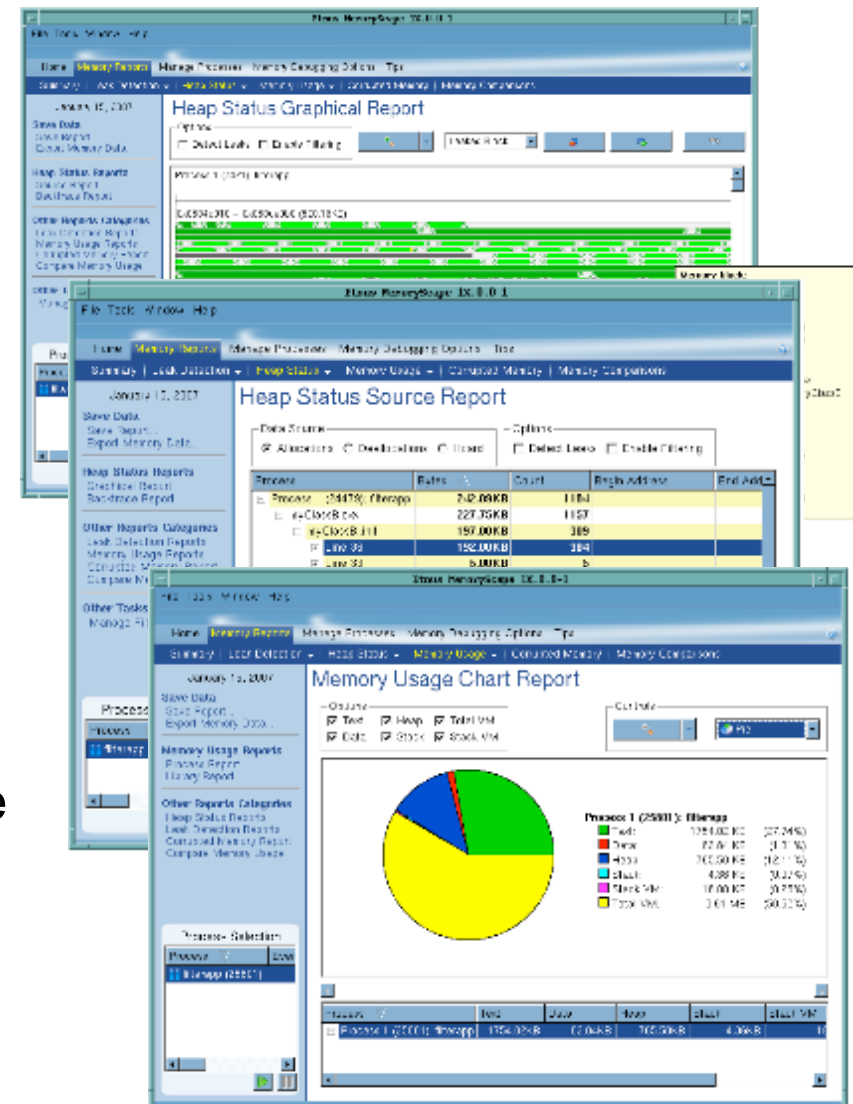
Session 1 Source Session 2 Source

Memory Usage Statistics



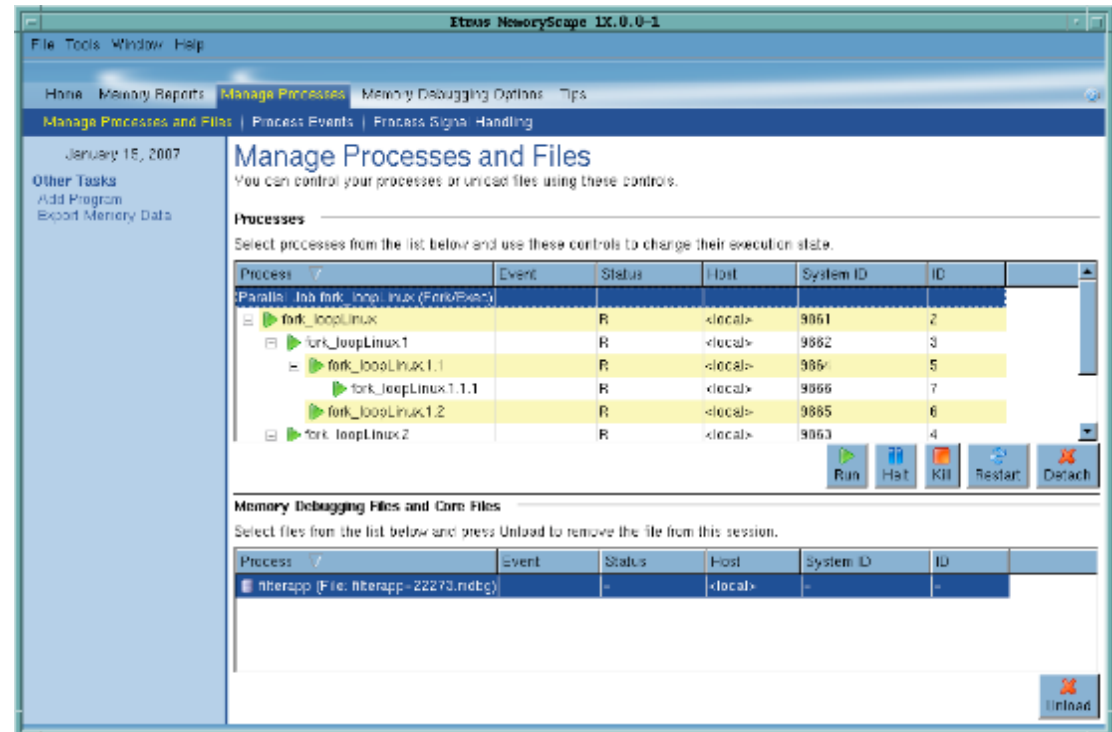
Memory Reports

- **Multiple Reports**
 - Memory Statistics
 - Interactive Graphical Display
 - Source Code Display
 - Backtrace Display
- **Allow the user to**
 - Monitor Program Memory Usage
 - Discover Allocation Layout
 - Look for Inefficient Allocation
 - Look for Memory Leaks



Multi-Process and Multi-Thread

- **Memory debug many processes at the same time**
 - MPI
 - Client-Server
 - Fork-Exec
 - Compare two runs
- **Remote applications**
- **Mutli-threaded applications**



MemoryScape 3.0

Improved Memory Hoarding support

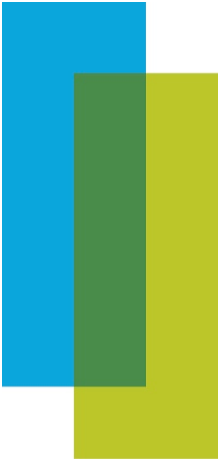
The use of memory hoarding in MemoryScape increases the risk of running out of available memory. MemoryScape now has the capability to manage this condition and alert you when you are at risk.

- **Hoard Low Memory Controls**
 - Automatically release hoarded memory when available memory gets low, allowing your program to run longer
- **Hoard Low Memory events**
 - MemoryScape can stop execution as notification that the hoard dropped below a particular threshold. This provides an indication that the program is getting close to running out of memory.
- **Hoard Low Memory scripting and CLI support**
 - TVScript
 - MemScript



Batch Debugging

tvscript



Batch Debugging

- **tvscript provides for unattended, straightforward TotalView batch debugging**
 - As an adjunct to interactive debugging
 - Usable whenever jobs need to be submitted or batched
 - Provides a tool more powerful and flexible than Printf-style debugging
 - Can be used to automate test/verify environments

Think of tvscript as “Printf on steroids”!



Batch Debugging

- Using tvscript, multiple debugging sessions can be run without the need for recompiling, unlike with printf
- A single compile is all that's needed, i.e.,
 - `gcc -g -o server-dbg server.c`
- **tvscript syntax:**
 - `tvscript [options] [filename] [-a program_args]`

tvscript

tvscript lets you define what events to act on, and what actions to take

- **Typical events**
 - Action_point
 - Any_memory_event
 - Guard_corruption
 - error
- **Typical actions**
 - Display_backtrace [-level *level-num*] [*num_levels*] [*options*]
 - List_leaks
 - Save_memory
 - Print [-slice {*slice_exp*} {*variable* | *exp*}
- tvscript also supports external script files, utilizing TCL within a CLI file allowing the generation of even more complex actions to events

tvscript

Example

- The following tells tvscript to report the contents of the *foreign_addr* structure each time the program gets to line 85

-create_actionpoint "#85=>print foreign_addr"

- Typical output blocks sample with tvscript:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Print
!
! Process:
!   ./server (Debugger Process ID:  1, System ID:  12110)
! Thread:
!   Debugger ID:  1.1, System ID:  3083946656
! Time Stamp:
!   06-26-2008 14:04:09
! Triggered from event:
!   actionpoint
! Results:
!   foreign_addr = {
!     sin_family = 0x0002 (2)
!     sin_port = 0x1fb6 (8118)
!     sin_addr = {
!       s_addr = 0x6658a8c0 (1717086400)
!     }
!     sin_zero = ""
!   }
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

tvscript

- Provides process, thread and timestamp information
- Provides a single output file, even with multiple processes
- Provides many event/action descriptions, including memory debugging events
- Supports external script files, utilizing TCL within a CLI file
- Allows the generation of even more complex actions



Remote Display Debugging

Easy, Secure and Fast

Remote Display Client

- **The Remote Display Client offers users the ability to easily set up and operate a TotalView debug session that is running on another system.**
- **TotalView Remote Display has two components:**
 - a Client, which runs on the remote system
 - a Server, which runs on any system supported by TotalView, “invisibly” manages the connections between the host and client
- **The Remote Display Client is available for:**
 - Linux x86
 - Linux x86-64
 - Windows XP
 - Windows Vista

Remote Display Client

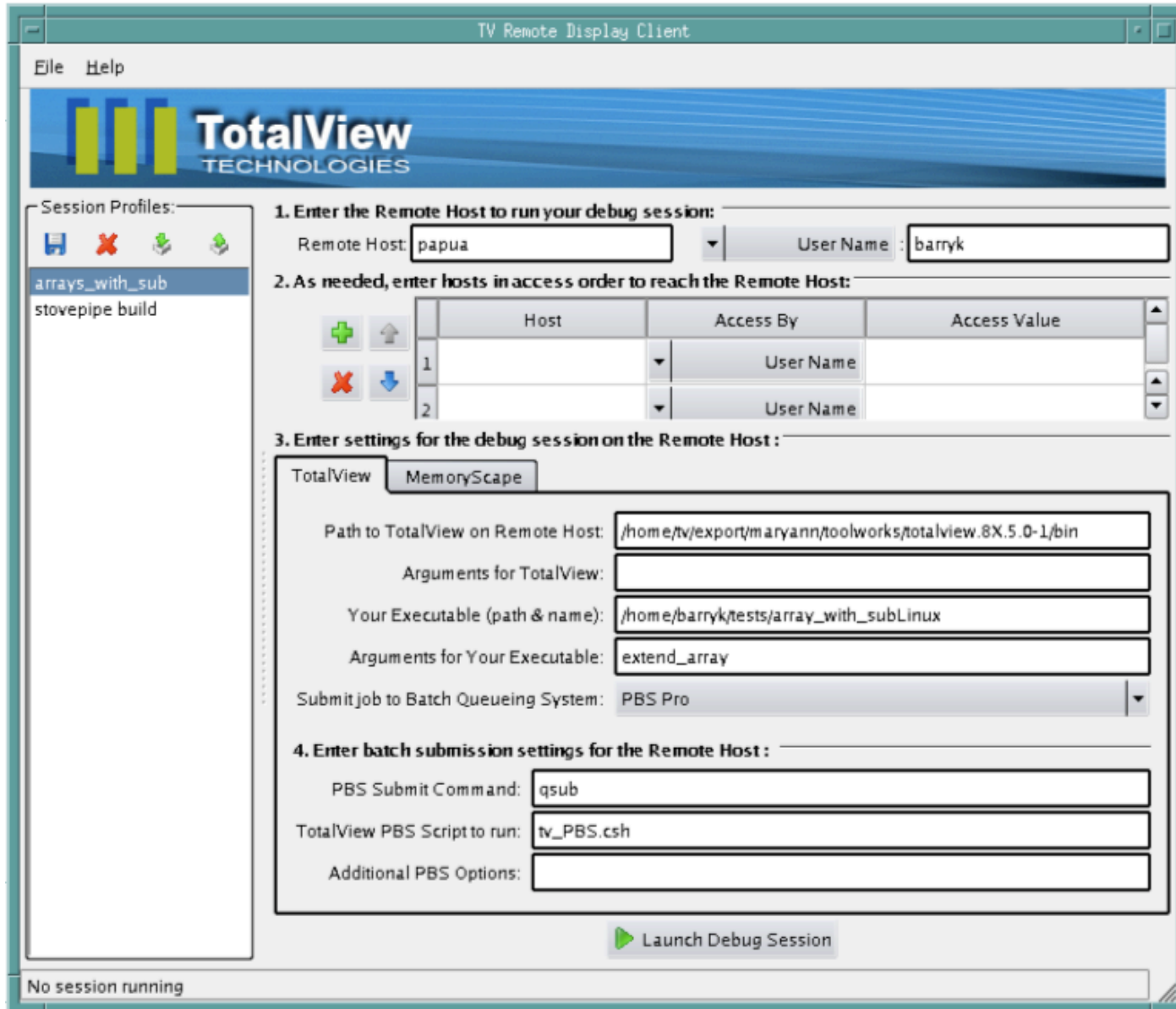
For Windows, TotalView provides a Set up Wizard



Remote Display Client

- **With the Remote Display Client window a user provides the information necessary for connectivity to the host, and saves the information in a profile.**
- **The information includes:**
 - User name, public key file, or other ssh options
 - The directory for TotalView or MemoryScape
 - The path name of the executable (full or relative to home directory)
 - For indirect connections the client provides for multiple intermediate host jumps, each by:
 - Host Name
 - Access type (User name, Public Key file, other SSH)
 - Access value
- **The Client also provides for submission of jobs to batch queuing systems PBS Pro and LoadLeveler**

Remote Display Client



The screenshot shows the 'TV Remote Display Client' window. It has a menu bar with 'File' and 'Help'. Below the menu bar is a banner with the TotalView Technologies logo. On the left is a 'Session Profiles' panel with icons for adding, deleting, and saving profiles, and a list of profiles: 'arrays_with_sub' and 'stovepipe build'. The main area contains four numbered steps for configuring a debug session. Step 1: 'Enter the Remote Host to run your debug session:' with fields for 'Remote Host' (papua) and 'User Name' (barryk). Step 2: 'As needed, enter hosts in access order to reach the Remote Host:' with a table for adding hosts. Step 3: 'Enter settings for the debug session on the Remote Host:' with tabs for 'TotalView' and 'MemoryScape', and fields for path, arguments, executable, and batch queueing system. Step 4: 'Enter batch submission settings for the Remote Host:' with fields for PBS submit command, script, and options. A 'Launch Debug Session' button is at the bottom right. A status bar at the bottom says 'No session running'.

TV Remote Display Client

File Help

TotalView
TECHNOLOGIES

Session Profiles:

arrays_with_sub
stovepipe build

1. Enter the Remote Host to run your debug session:

Remote Host: papua User Name: barryk

2. As needed, enter hosts in access order to reach the Remote Host:

| | Host | Access By | Access Value |
|---|------|-----------|--------------|
| 1 | | User Name | |
| 2 | | User Name | |

3. Enter settings for the debug session on the Remote Host:

TotalView MemoryScape

Path to TotalView on Remote Host: /home/tv/export/maryann/toolworks/totalview.8X.5.0-1/bin

Arguments for TotalView:

Your Executable (path & name): /home/barryk/tests/array_with_subLinux

Arguments for Your Executable: extend_array

Submit job to Batch Queueing System: PBS Pro

4. Enter batch submission settings for the Remote Host:

PBS Submit Command: qsub

TotalView PBS Script to run: tv_PBS.csh

Additional PBS Options:

Launch Debug Session

No session running

Remote Display

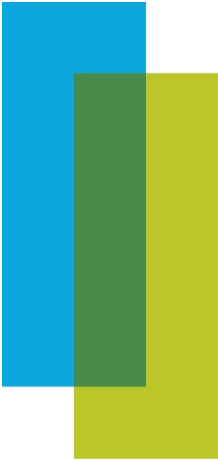
Remote Display

Session Profile Management

- Profiles are saved with profile names
- Multiple profiles can be generated for various environments
- Profiles can be exported and shared
- Shared Profile files can be imported and shared by other users

Remote Display Security

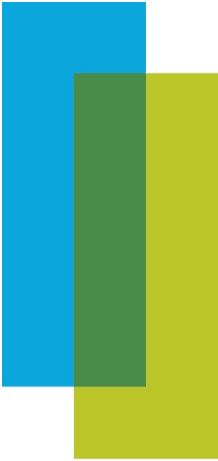
- Remote Display uses SSH
- Remote Display Server allows only RFB (Remote Frame Buffer) connections to and from the host
- No incoming access to the Server is allowed
- The Server can only connect back to the Viewer via SSH
- Only one Viewer connection is allowed to the Server
- As Remote Display connects to systems, the user is prompted for password information as required
 - Note: ssh and X Windows are required



Thanks!

QUESTIONS?

totalviewtech.com

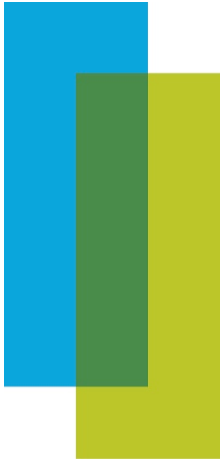


TotalView Customer Support

support@totalviewtech.com

Memory Debugging Discussions at:

debugopedia.com/



TotalView Customer Support

support@totalviewtech.com

Starting TotalView on ALCF Bluegene



- Use `ssh -Y` (or `-X`) to login so that an X11 tunnel is created
- Add `'+totalview'` to your `$HOME/.softenvrc` file
- To submit a job for debugging use `isub`:

Usage:

```
isub <qsub_args> --run totalview <tv_args> mpirun -a <mpi_args>
```

e.g.:

```
isub -t 30 -n 32 -A myproject --run totalview mpirun -a -np 32 a.out
```

- TotalView will start when job is allocated
- You start of debugging 'mpirun'
 - Typically the first thing you want to do is hit the “GO” button.
- Be patient while the partitions boot, TotalView will tell you when the application has been halted
 - All processes will be halted after executing exactly one instruction.